Set Automata

Klaus-Jörn Lange and Klaus Reinhardt *

Wilhelm-Schickhard Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Germany
e-mail: {lange,reinhard}@informatik.uni-tuebingen.de

Abstract

In order to characterize certain formal languages capturing some syntactic aspects of higher programming languages we introduce automata with the storage type *set*, called *set automata*. We show the corresponding language class to have an *NP*-complete membership problem and a decidable emptiness problem.

1 Introduction

One of the main issues of formal languages is to model the syntax of higher programming languages or parts of it. This leads for instance to grammars generating parenthesis expressions or representing recursive structures. A limit of these approaches has often been the existence of certain constraints which have to be fullfilled by the syntactical objects. A very typical example is the problem of declaring variables in programs written in languages like PASCAL. In its most simple form without concerning recursive or block structures it can be formulated as: *Each variable used in a program has to be declared exactly once!* Formally, this means that there exists a mapping from the set of all occurrences of variables being used in a program to the set of all declarations of variables in that program. This condition of being a mapping can be split into the two subconditions *totality on the domain* and *uniqueness on the range*, i.e. first, that for each element of the domain (here the set of all occurences of variables) there is a corresponding element in the range (here the corresponding declaration), and second, that for no element in the domain there are two or more corresponding elements in the range (which here means that no variable is declared twice or more). These two conditions can be expressed in terms of the following two formal languages:

$$L_{td} = \left\{ x_1 c x_2 \cdots c x_n d y_1 c y_2 \cdots c y_m \;\middle|\; \begin{array}{l} m,n \geq 1, x_i, y_j \in \{a,b\}^*, \\ \bigwedge_{1\leq i\leq n} \bigvee_{1\leq j\leq m} x_i = y_j \end{array} \right\}$$

$$\text{and } L_{ur} = \left\{ y_1 c y_2 \cdots c y_m \;\middle|\; m \geq 1, y_i \in \{a,b\}^*, \bigwedge_{i\neq j} y_i \neq y_j \right\}.$$

* This research has been supported by the DFG Project La 618/3-1 KOMET.

The set L_{ur} is also known as the problem of *Element Distinctness*. On the one hand, being elements of AC^0 both languages have a very simple membership problem. On the other hand, capturing them in terms of formal languages, that is describing by grammars or automata seems to be quite difficult. Fischer,addressing the problem of well-declaration, was able to construct an indexed grammar generating L_{td} ([2]). In fact, L_{td} is even an element of $ETOL$. But both these grammar types have in general an NP-complete membership problem.

It is the aim of this work to represent L_{ur} by an automaton type sharing typical properties of formal languages. In particular, we look for an automaton type generating a class of languages constructively closed under morphisms, inverse morphisms, and intersection with regular sets and having a decidable emptiness problem enabling the existence of pumping or copying results.

For this reason we introduce *set automata* which are automata with access to a datastructure of type set. A set automaton has a read-only one-way input tape, a finite set of states, and write-only one-way output tape, called the *query tape*. It can perform three different kind of steps: reading the next symbol from the input tape, writing another output symbol on the query tape, and *asking a question*, which means that the word currently written on the query tape (called the query) is compared with all queries asked previously in this computation. If it coincides with a previous query, the *answer* is yes, otherwise no. The answer is given to the automaton in terms of its finite states. Simultaneously, the query tape is erased.

It turns out that the class $\mathcal{SET}$ of all *set languages*, that is of all languages accepted by set automata, forms a trio, i.e. is closed under morphisms, inverse morphisms , and intersection with regular sets and that these closure properties are constructive. As our first main result of this work we show the decidability of the emptiness problem for set automata. Our second main result proves the NP-completeness of the membership problem even if we restrict the automaton to accept only if all queries have been answered negatively.

We assume the reader to be aquainted with the basic notions and results of formal language theory as they are contained for instance in the books of Hopcroft and Ullman or by Salomaa ([3,4]). Throughout of this paper we denote the empty word by λ and the cardinality of a set M by $|M|$.

For lack of space we can only give indications for most of our constructions and proofs.

2 Set automata

This section starts with the formal definitions of set automata and of set languages. After some results concerning formal language aspects it continues with questions of decidability and complexity.

2.1 Definition of the set-automaton

A set-automaton has a nondeterministic finite control, a one-way input-tape and a one-way output-tape, which we call the *query-tape*. There are three kinds

of elementary operations: The reading of an input symbol (δ_r–transitions), the writing of a symbol to the query-tape (δ_w–transitions) and the posing of a query (δ_q–transitions). This posing of a query means that the following state depends on whether the word on the query-tape is in the set of words, which have been asked before. Simultaneaously the word is included in this set and the query-tape is cleared. Accordingly we have two special kinds of states: the *query-states* and the (positive and negative) *answer-states*. For technical reasons, we regard both the initial state and the final states as answer-states.

Definition 1. A *set-automaton* is a 8-tuple $A = (Z, \Sigma, \Gamma, \delta, Q, S, E, z_0)$, where

- Z is a finite set of *states*
- Σ is a finite *input-alphabet*,
- Γ is a finite *query-alphabet*,
- $Q \subseteq Z$ is a set of *query-states*
- $S \subseteq Z \setminus Q$ is a set of *answer-states*,
- $E \subseteq S$ is a set of *accepting states*,
- $z_0 \in S$ is an *initial state* and
- $\delta = \delta_r \cup \delta_b \cup \delta_q$ with

  $$\delta_r \subset (Z \setminus Q) \times \Sigma \times (Z \setminus S),\ \delta_w \subset (Z \setminus Q) \times \Gamma \times (Z \setminus S),\ \delta_q \subset Q \times S \times S$$

  is a finite set of *transitions*.

To describe calculations of a set-automaton, we have to define the notion of a *configuration*. A configuration consists of the actual state, the rest of the input-tape, the part of the query-tape written so far and the set of the priviously asked queries.

Definition 2. Let $A = (Z, \Sigma, \Gamma, \delta, Q, E, S, z_0)$ be a set-automaton.

a.) A *configuration* of A is an element of $Z \times \Sigma^* \times \Gamma^* \times 2^{\Gamma^*}$.

b.) A configuration (z', v', α', M') of A is the *successor* of a configuration (z, v, α, M) (written as $(z, v, \alpha, M) \vdash (z', v', \alpha', M')$) if and only if one of the following holds:

- **Reading an input-symbol:** There is a transition $(z, a, z') \in \delta_r$ and it holds $v = av'$, $\alpha = \alpha'$ and $M = M'$,
- **Writing to the query-tape:** There is a transition $(z, x, z') \in \delta_w$ and it holds $\alpha x = \alpha'$, $v = v'$ and $M = M'$, or
- **Asking a question:** There is a transition $(z, z_+, z_-) \in \delta_q$ and it holds either (positive answer) $\alpha \in M$, $M = M'$ and $z' = z_+$ or (negative answer) $\alpha \notin M$, $M' = M \cup \{\alpha\}$ and $z' = z_-$. In both cases we have $v = v'$ and $\alpha' = \lambda$.

We use $\vdash^*$ for the transitive closure of the relation $\vdash$.

c.) The *language accepted by* A is

$$L(A) = \left\{ w \in \Sigma^* \;\middle|\; \bigvee_{z \in E} \bigvee_{\alpha \in \Gamma^*} \bigvee_{M \subset \Gamma^*} (z_0, w, \lambda, \emptyset) \vdash^* (z, \lambda, \alpha, M) \right\}.$$

In the following we call a laguage accepted by a set automaton a *set language*. The class of all set languages is denoted by $\mathcal{SET}$.

2.2 Formal laguage properties

From the description of set-languages based on automata, we get directly some closure properties, which we state here without proof. We mention in passing that they are constructive in the sense that given descripitons of a set automaton A and of a morphism h or of a regular set R, the description of a set automaton accepting $h(L(A))$, $h^{-1}(L(A))$, or $L(A) \cap R$ is computable by a recursive function.

Theorem 3. *The family $\mathcal{SET}$ of set languages forms a TRIO, which means, they are closed under (possibly erasing) homomorphisms, inverse homomorphisms, and intersections with regular languages.*

In the following we present a generator playing the same rôle for $\mathcal{SET}$ that the Dyck sets have for the context-free languages. Just as the Dyck sets reflect the proper use of a push down store in terms of matching push- and pop-moves, we consider the set of all correct set computations with matching positive and negative queries to a set built by these queries.

Definition 4. The language of correct set computations is defined by:

$$L_{+,-} = \left\{ y_1\delta_1 c y_2\delta_2 \cdots c y_m\delta_m \;\middle|\; \begin{array}{c} m \geq 1, y_i \in \{a,b\}^*, \delta_i \in \{+,-\}, \\ \bigwedge_{1\leq i\leq m}(\delta_i = +) \Leftrightarrow \left(\bigvee_{j<i} y_j = y_i\right) \end{array} \right\}$$

Let us observe, that the original language L_{ur} as considered in the introduction corresponds to the subset of those words in $L_{+,-}$ where no $+$ appears, this means that no question is answered positive. By converting positive and negative answers we could view L_{ur} as conjunctive restriction of $L_{+,-}$, if we regard set-automata as oracle machines with set oracles.

Theorem 5. *$L_{+,-}$ is a trio-generator of the class of set-languages, this means for every set-language L there are two homomorphisms f and g and a regular language R such that $L = f(g^{-1}(L_{+,-}) \cap R)$.*

Proof: Obviously, $L_{+,-}$ is a set language. Thus, Theorem 3 implies the inclusion $\mathrm{TRIO}(L_{+,-}) \subseteq \mathcal{SET}$. Now, it suffices to show that every set-language L can be reduced to $L_{+,-}$ by a finite transducer. Such a finite transducer simulates the set-automaton accepting L by writing the symbols which it writes to the query tape to the output tape. If the set-automaton enters the query state, the finite transducer nondeterministically guesses the answer and either it writes a $+$ to the output tape and continues the simulation in the positive answer state or it writes a $-$ to the output tape and continues the simulation in the negative answer state. Obviously, the nondeteministic transducer can output an element of $L_{+,-}$ if and only if its input word belongs to L. ∎

It is easy to see that $\mathcal{SET}$ is closed under union and catenation, as well. Closure under union is provided by the machine's nondeterminism. The simple idea to prove closure under catenation is to use two disjoint query-alpabets when simulating the two underlying set automata. But this method does not work for

the operation of Kleene's closure. We leave it as an open question whether $\mathcal{SET}$ is closed under Kleene's star which would make $\mathcal{SET}$ a full AFL. A possible counterexample is the language $(L_{+,-}\,\$\,)^*$, i.e. the marked Kleene's closure of $L_{+,-}$.

3 Decidability and Complexity

Theorem 6. *The emptyness problem for set automata is decidable.*

Proof: The gist of the following construction is to separate in a computation of a set automaton the query transitions from the pure reading and writing steps. The later, without access to the storage medium, can be described by rational transductions.

Let $A = (Z, \Sigma, \Gamma, \delta, Q, E, S, z_0)$ be a set-automaton. Let in the following m be the size of $S \times Q$, i.e. $m := |S| \cdot |Q|$. Obviously we can regard A without the δ_q-transitions as a finite automaton with input and output, calculating a rational transduction $\tau_A \subset Z \times \Sigma^* \times \Gamma^* \times Z$ (see e.g. [1]). The relation τ_A consists of exactly those tuples (z, w, α, z') such that there is a path from z to z' of δ_r- and δ_w-transitions reading w and writing α. For every answer-state $s \in S$, and every query-state $q \in Q$, the set

$$R_{s,q} = \left\{ \alpha \in \Gamma^* \,\middle|\, \bigvee_{w \in \Sigma^*} (s, w, \alpha, q) \in \tau_A \right\}$$

is regular. Therefor for every subset $N \subseteq S \times Q$ the following set is regular as well:

$$R_N = \left\{ \alpha \in \Gamma^* \,\middle|\, \bigwedge_{(s,q) \in S \times Q} \alpha \in R_{s,q} \Leftrightarrow (s,q) \in N \right\}.$$

The 2^m (possibly empty) sets $R_N, N \subseteq S \times Q$, form a partition of Γ^*.

Let $F = \{N \subseteq S \times Q \,|R_N \text{ is finite}\}$ and $I := (S \times Q) \setminus F$. We now define the set of *global states*:

$$K := S \times \{0, \infty\}^{|I|} \times \prod_{N \in F} \{0, 1, \cdots, |R_N|\}.$$

Observe, that K is finite. We let the *global start state* be $(z_0, \overbrace{0, 0, \cdots, 0}^{m-times})$. A *global end state* is a global state $(s, i_1, \cdots, i_m)$ with $s \in E$. If the set-automaton M is in an answer state s after asking a question, then the global state not only contains the state s but also for every $N \subseteq S \times Q$ the number of previously asked words which are in R_N. For the case of an infinite R_N, we just store whether no $(i_N = 0)$ or at lest one $(i_N = \infty)$ word of R_N was asked.

We now regard the elements of K as nodes of a directed graph having the following edges:

$$(s, i_1, i_2, \cdots, i_m) \longrightarrow (s', i'_1, i'_2, \cdots, i'_m)$$

if and only if there is a $(q, s_1, s_2) \in \delta_q$ with:

Case 1: 'Query with positive answer'

$$s' = s_1 \wedge \bigwedge_{N \subseteq S \times Q} (i'_N = i_N \wedge [(s,q) \in N \Rightarrow i_N \neq 0])$$

or

Case 2: 'Query with negative answer'

$$s' = s_2 \wedge \bigwedge_{N \subseteq S \times Q} ((s,q) \notin N \Rightarrow i'_N = i_N) \wedge$$

$$\bigwedge_{N \subseteq S \times Q} [((s,q) \in N \wedge N \in I) \Rightarrow i'_N = \infty] \wedge$$

$$\bigwedge_{N \subseteq S \times Q} [((s,q) \in N \wedge N \in F) \Rightarrow (i_N < |R_N| \wedge i'_N = i_N + 1)] .$$

The idea behind this construction is that we represent valid computations of A by paths in a graph. Sequences of reading and writing steps are captured by the sets $R_{s,q}$ and R_N. Query steps leading from s via q to s_1 or S_2 lead to an update of the counters. Positively answered queries, for instance, mean that all counters remain unchanged since nothing is inserted in the history of asked questions, but that for each set N containig (s,q) the counter for R_N must be nonzero, since there must exist a query leading from s to q which was posed previously. Hence, there is an accepting calculation of A if and only if there is a way in the graph of global states from the global start state to a global end state. The decidability follows from the finiteness of the set K. ■

Because $w \in L(A)$ holds if and only if $L(A) \cap \{w\}$ is not empty, we get as a consequence of the effective closure of the set-languages under intersection with regular sets:

Corollary 7. *The word problem for set-languages is decidable.*

Theorem 8. *There exist NP-complete set languages. Hence the membership question for set languages is NP-hard.*

Proof: From the coding F of a boolean formula in conjunctive normal form we can compute (in logarithmic space) the word $V\$F$ where V is a list of all variables occuring in F. A set automaton A first reads the partial word V and guesses for every variable x_i its value by either posing the query x_i1 or x_i0, which, of course, are all answered negatively. Afterwards A reads the formula F and guesses for every clause the fulfilling literal. A proves the correctness by posing queries of the form x_i1 or x_i0, which now all have to be answered positive. In this case A accepts, otherwhise A rejects. ■

Theorem 9. *The word problem for set automata is NP-complete.*

Proof: Because of Theorem 8 it suffices to show that if the set automaton A accepts an input x, then it can do so making only $p(|x|)$ steps for a polynomial p. Then a nondeterministic polynomial timebounded Turing machine can simulate A.

We use the terminology and parameters of Theorem 6. Let $n = |x|$ be the length of the input x. Continuing the construction of Theorem 6 let $g := |K|$ be the number of global states. In the following, we distinguish in a computation of A between *special queries* which are generated by both reading and writing transitions and *normal queries* which are built using writing steps, only. During an accepting calculation of A on x, A can ask at most n special queries. For each positively answered special query asking the word α, we regard that query which asked this query word α the first time as a special query, as well. This means we have to consider at most $2n$ special queries. If the calculation of A on x reaches a global state twice without asking a special query inbetween, then this part of the calculation can be skipped and later positive queries related to negative queries in this part can be replaced by an earlier query in the same R_N. (Such a query must already exist in the set, otherwise the global state would have changed.) In this way we can reduce an accepting calculation of A on x to making at most $2gn = O(n)$ queries.

Let $(\beta_i)_{1 \leq i \leq m}$ for some $m = O(n)$ be the sequence of queries asked by A during an arbitrary, but fixed successful computation on input x. Now let $\Omega := \{\beta_i | 1 \leq i \leq n\}$ be the set of different query words. For each $\gamma \in \Omega$ let $index(\gamma)$ be the index of the first query asking for γ which inserted γ into Ω and had been answered negatively, i.e.: $index(\gamma) := \mathrm{MIN}\{i | \beta_i = \gamma\}$. We now decompose each query β_i into pieces such that no input symbol has been read while writing a piece. Then we decompose each $\gamma \in \Omega$ into $\gamma = \gamma_0 \gamma_1 \cdots \gamma_t$ by overlaying the decompositions of all queries β_i with $\beta_i = \gamma$. Since there are at most n input symbols to be read, we have $t = O(n)$.

For each $z, z' \in Z$ and for each $M \subseteq Z \times Z$ we consider the sets:

$$T_{z,z'} = \{\alpha \in \Gamma^* \,|(z, \lambda, \alpha, z') \in \tau_A\}$$

and

$$T_M = \bigcap_{(z,z') \in M} T_{z,z'}.$$

We partition $Z \times Z$ into the two parts $FIN := \{M \subseteq Z \times Z | T_M \text{ is finite}\}$ and its complement $INF := \{M \subseteq Z \times Z | T_M \text{ is infinite}\}$. We now let c_1 be the length of the longest word in some finite T_M, i.e.: $c_1 := \mathrm{MAX}_{M \in FIN} \mathrm{MAX}_{v \in T_M} |v|$. Further on, we let c_2 be the minimal integer such that the minimal period of the word length of each infinite T_M is a divisor of c_2, i.e. there exists an wrt. c_2 minimal positive integer c_3 such that for all $M \in INF$ and for all $v \in T_M$ with $|v| > c_3$ there exist words $v', v'' \in T_M$ such that $|v'| = |v| - c_2$ and $|v''| = |v| + c_2$. We now let c be the maximum of c_1, c_2, and c_3.

The decomposition $\gamma = \gamma_0 \gamma_1 \cdots \gamma_t$ induced by the decomposition of some queries $\beta_{i_1}, \cdots, \beta_{i_r}$ means for each γ_j the existence of pairs $(z_1, z_1'), \cdots, (z_r, z_r')$ such that γ_j has to be a member of $T_{z_i, z_i'}$ for each $1 \leq i \leq r$. For each γ_j let

M_j be the set of these pairs of states. That is, the decomposition of γ asks for $\gamma_j \in T_{M_j}$ for each j. If now $M_j \in FIN$ for each $0 \leq j \leq t$ then we know $|\gamma| \leq (t+1) \cdot c_1 = O(n)$. Now let us assume, that $M_j \in INF$ for some $0 \leq j \leq t$. By the definition of c_2 we can replace each γ_j for which T_{M_j} is infinite by some γ'_j of length $|\gamma'_j| \leq c_3$. This would result in a query candidate γ' of length $|\gamma'| \leq (t+1) \cdot c = O(n)$. The problem is that this modified query could interfere with other modified or unmodified elements of Ω. To avoid this we will increase the length of γ' again such that we can be sure that all pumped queries have different length. So let $i := index(\gamma)$. We now simply increase the length of γ' in steps of length c_2 until we have $(n+j) \cdot c < |\gamma'| \leq (n+j+1) \cdot c$. This prevents any interference of the modified queries with each other while keeping $|\gamma'| \leq (n + index(\gamma) + 1) \cdot c = O(n)$.

Thus we know, that if there is an accepting computation, there is also one with $O(n)$ queries of length $O(n)$ which can be guessed in time bounded polynimial in n. ■

Theorem 10. *The word problem for set languages accepted by set automata with only negative answers which is the trio generated by L_{ur} is NP-complete.*

Proof: Let F be the coding of a formula in KNF. By a log-reduction we can construct the word $x_1\$x_1\$\$x_1\$\$\$...x_1\$^{k_1}x_2\$...x_n\$^{k_n}F'$,where $k_i = \#_{x_i}(F)$ is the number of occurances of the variable x_i in F and F' is formula F modified in a way that the jth occurence of variable x_i is coded by $x_i\#^j$. While reading the first part of the input, a set automaton A guesses an assignement for every variable x_i and either puts all the strings $x_i1, \cdots, x_i1^{k_i}$ into the set if the assignement is false or all the strings $x_i0, \cdots, x_i0^{k_i}$ if the assignement is true. While reading F', for every clause A guesses the fulfilling literal and verifies the correctness by puting the query x_i1^j or x_i0^j, where j is given by the coding. So if and only if for example x_i is true, then it is possible to ask negatively for x_i1^j, which is still not in the set. The automaton A only accepts if all answers to queries are negative. ■

Finally, we mention without proof that in the deterministic case we get a P-complete word problem:

Theorem 11. *The word problem for deterministic set automata is P-complete.*

4 Discussion

In designing the notion of a set automaton it was essential to restrict the model severely in order to end up with a decidable emptiness problem. On the other hand, this led to a rather 'uncomfortable' model. It is still open, whether $\mathcal{SET}$ is closed under Kleene's star. One possibility to gain a full AFL is to give set automata the possibility of emptying the sets of asked queries. In this way we come to the trio generated by the set $(L_{+,-}\$)^*$. Here the symbol \$ indicates the operation of emptying the set of old queries. This trio is obviously closed under the $*$−operation while still having a decidable emptiness problem.

On the other hand, with an *NP*-complete membership problem set automata are of a rather high complexity. If we enhance the model by allowing to put a query without puting the word into the set of previous questions, the complexity results remain the same. But if we also allow to delete the word on the query tape from the set, then the infinite sets R_N constructed in the proof of Theorem 6 behave like counters without zero test which means that the word and the emptyness problem become equivalent to the reachability problem in Petri nets.

Things change if we allow the set automata to put a query without emtying the query tape (and afterwards continue the query word). In this case the word and the emptyness problem become undecidable since it is possible to simulate automata with two counters by set automata of the extended type, even if there is only a single query symbol: The contents n_c of counter $c \in \{0,1\}$ is represented by the set containing the words $a^c, a^{c+4}, ...a^{c+(o_c)4}$ and $a^{c+2}, a^{c+6}, ...a^{c+(o_c+n_c)4+2}$. To perform a zero test of counter c, the set automaton asks all queries $a^c, a^{c+2}, ...a^{c+(o_c+1)4+2}$, where exactly the last two must be answerded negative. To perform a decrement of counter c, the set automaton asks all queries $a^c, a^{c+2}, ...a^{c+(o_c+1)4+2}$, where exactly the query $a^{c+(o_c+1)4}$ must be answerded negative. To perform a increment of counter c, the set automaton asks all queries $a^{c+2}, a^{c+6}, ...a^{c+(o_c+n_c)4+6}$, where only the last query must be answerded negative.

References

1. J. Berstel. *Transductions and Context-Free Languages.* Teubner Verlag, Stuttgart, 1979.
2. M.J. Fischer. Grammars with macro-like production. Ph.d. thesis, Harvard Univ., 1968.
3. J. Hopcroft and J. Ullman. *Introduction to Automata Theory, Language, and Computation.* Addison-Wesley, Reading Mass., 1979.
4. Arto Salomaa. *Formale Sprachen.* Springer Verlag, 1974.