Complexity of Automata with Abstract Storages [1]

Jürgen Dassow
Technische Universität Magdeburg
Fakultät für Mathematik
Magdeburg, Germany

Klaus-Jörn Lange
Technische Universität München
Institut für Informatik
München, Germany

Abstract. The complexities of problems related to automata with storages, an abstract automaton model introduced by Engelfriet, are analyzed. In particular, we show a close relationship between the word problem of two-way automata and the emptiness problem of one-way automata.

1. Introduction

In this paper we want to build a further bridge between the areas of formal languages and of complexity. There are a lot of characterization results of complexity classes with the help of formal language predicates and operations (see e.g. [14] , [11]). A very typical result of this kind is the complexity theoretical equivalence of the word problem of certain classes of two-way automata and the emptiness problem of the corresponding class of one-way automata. This phenomenon was observed for a large variety of automata (see [3] , [9] , and [10]). Another result of this kind is the complexity theoretical interpretation of theorems of Chomsky/Schutzenberger and Greibach. Here we intend to derive these results on a very abstract level. Approaches

[1]This work was supported by SFB 342, Teilprojekt A4 of the Deutsche Forschungsgemeinschaft

like that were given before within the framework of balloon automata and abstract families of languages (see [6]). A more suitable formalism for this aim was exhibited by Engelfriet which introduced *automata with storages* (see [4] , [5]). The advantage of this approach is a very clear distinction between automaton and storage. To show this advantage with respect to complexity theory is one of our aims.
This paper is organized as follows. In Section 2 we give the necessary definitions and present some basic facts, e.g. a normal form for automata with storages. Section 3 contains the relation between membership problem for two-way automata and emptiness problem for one-way automata of the same type. In Section 4 we show the existence of hardest languages and analoga of Dyck and Greibach languages.

2. Definitions and preliminary results

Throughout this paper we assume that the reader is familiar with the basic concepts of formal language and complexity theory as contained in [8] and [17] .
We start with some notation.
For a set P of predicates over a set C we denote by $BE(P)$ the set of Boolean expressions over P using the operators $\wedge, \vee, \neg$, $true$, and $false$. For $b \in BE(P)$ and $c \in C$ the value of $b(c)$ is defined in the obvious way.
λ denotes the empty word.
We call a mapping $f : X^* \longrightarrow Y^*$, where X and Y are finite alphabets, computable within logarithmic space (resp. polynomial time), if there is a Turing-machine M which, started with some $v \in X^*$ on its read only input tape, computes and prints $f(v)$ on its write only output tape, consuming no more than $O(log(|v|))$ space on its work tapes (resp. performing no more than $p(|v|)$ steps for some polynomial p).
A set $L \subseteq X^*$ is many-one LOG-*reducible* to a set $M \in Y^*$, denoted by $L \leq_m^L M$ if there exists a mapping $f : X^* \longrightarrow Y^*$ computable within logarithmic space such that $v \in L$ if and only if $f(v) \in M$ for all $v \in X^*$. The LOG-*closure* of a class $\mathbf{X}$ of languages is defined as

$$LOG(\mathbf{A}) = \{L : \text{there is } M \in \mathbf{A} \text{ with } L \leq_m^L M \}.$$

If $\mathbf{A}$ consists of a single element L_0 we write $LOG(L_0)$ instead of $LOG(\mathbf{A})$.
A class $\mathbf{A}$ is complete for a class $\mathbf{B}$ if $\mathbf{A} \subseteq \mathbf{B} \subseteq LOG(\mathbf{A})$.
We now give the definition of automata with storages and related notions, which differ slightly from those given in [4] by using of final configurations;

however, we cover that concepts.

A *storage type* is a quintuple

$$S = (C, P, I, C_{in}, C_{fin})$$

where C is a set of *configurations*, P is the finite set of *predicates* over C, I is the finite set of *instructions*, and $C_{in} \subseteq C$ and $C_{fin} \subseteq C$ are the sets of *initial* and *final* configurations, respectively.

Example 1. For a fixed set Γ of push-down symbols, we define the storage type $PD(\Gamma)$ by

$$C = \Gamma^*,\ C_{in} = \Gamma,\ C_{fin} = C,$$
$$P = \{top_x : x \in \Gamma\} \cup \{bottom\},$$
$$I = \{push_x : x \in \Gamma\} \cup \{stay_x : x \in \Gamma\} \cup \{pop\}$$

and the usual meaning

$$top_x(w) = true \text{ if and only if } w = w'x \text{ for some } w',$$
$$bottom(w) = true \text{ if and only if } w = \lambda,$$
$$push_x(w) = wx,$$
$$stay_x(wx) = wx \text{ and } stay_x(wy) \text{ is undefined for } y \neq x, y \in \Gamma,$$
$$pop(wx) = w \text{ for } x \in \Gamma.$$

In the same way it is possible to define storage types corresponding to stacks, nested stacks, non-erasing stacks and checking steps. However, it is not possible to characterize restricted push-down arrays of counters introduced by Rozenberg (see [15]), although some the results given below are also valid for this class (see [12]).

The crucial notion of this paper is the following.

A *two-way S-automaton* is a 6-tuple

$$A = (Q, X, \delta, q_{in}, c_{in}, F)$$

where
- Q is a finite set (of *states*),
- X is the *(input) alphabet*,
- $q_{in} \in Q$ is the *initial* state,
- $c_{in} \in C_{in}$ is the initial storage configuration,
- $F \subseteq Q$ is the set of *accepting* states,
- δ is a finite subset of $Q \times X \times BE(P) \times Q \times \{L, N, R\} \times I^*$.

L, R, N, are interpreted as the moves of the reading head to the left or to the right or as no move, respectively. The elements of δ are called *transitions*.

By $< A >$ we denote a description of A as a word over some alphabet where

we assume that these description contains also P and I. The size of A is the length of $< A >$.

An *instantaneous description* of A is an element $(q, w_1\#w_2, c)$ of $Q\times X^*\#X^*\times C$. The intuitive meaning is that A is in the state q, w_1w_2 is the word on the input tape, the reading head reads the first letter of w_2 , and c is the storage configuration. For two instantaneous descriptions $t_1 = (q, w_1y\#xw_2, c)$ and $t_2 = (q', w_1'\#w_2', c')$, we say that t_1 derives directly t_2, written as $t_1 \vdash_A t_2$, if there exist a transition $(q, x, \beta, q', r, \gamma) \in \delta$ such that $\beta(c)$ is true, $c' = \gamma(c)$, and $w_1' = w_1yx, w_2' = w_2$ for $r = R, x \in X$ or $w_1' = w_1y, w_2' = xw_2$ for $r = N, x \in X$ or $w_1' = w_1, w_2' = yxw_2$ for $r = L, x \in X$.

Let $\vdash_A^*$ be the reflexive and transitive closure of $\vdash_A$. The language $L(A)$ accepted by A is defined as

$$L(A, S) = \{z : (q_{in}, \#z, c_{in}) \vdash_A^* (q, z_1\#z_2, c) \text{ for some } q \in F \text{ and } c \in C_{fin}\}.$$

We say that the S-automaton is *deterministic* if, for any $p \in Q$, $x \in X$ and $c \in C$, there is at most one transition $(p, x, b, q, r, f) \in \delta$ with $b(c) = true$. We say that the S-automaton is a *one-way* automaton, if the head cannot move to the left. Obviously, the λ-transitions usually permitted for one-way automata can be covered by transitions with no move of the head.

In addition, we consider *augmented* S-automata which are equipped with a logarithmically bounded worktape. Therefore an augmented S-automaton has additional components, which represent the alphabet of an additional worktape and the initial symbol at the worktape; the transitions give also the information on the reading, writing and move of the head scanning the worktape, and the workspace used for the computation on the input z is bounded by $O(log(|z|))$.

We say that the S-automaton A is in *normal form* iff δ is a subset of $Q \times X \times P' \times Q \times \{L, R, N\} \times (I \cup \{\lambda\}$ where $P' = P \cup \{\neg p : p \in P\} \cup \{true\}$.

Let S and $\mathbf{S}$ be a storage type and a class of storage types, respectively. Then, for $x \in \{1, 2\}$, we denote by **x-N**S**A** the family of all languages accepted by x-way S-automata, and we set

$$\textbf{x-NSA} = \bigcup_{S\in\mathbf{S}} \textbf{x-}S\textbf{A}.$$

If the automata are deterministic, then we use the letter **D** instead of **N** in the notation of the family of accepted languages. If the automata are augmented, then we add the letter **A** after the **N** or **D**. Thus, for example, **1-DA**S**A** denotes the family of languages accepted by one-way augmented deterministic S-automata.

Example 1. (continued) It is easy to see that a $PD(\Gamma)$-automaton $A = (Q, X, \delta, q_{in}, c_{in}, F)$ is a usual push-down automaton. Therefore, $\#(\Gamma) \geq 2$,

1-N$PD(\Gamma)$**A** is the family of all context-free languages.
Let

$$\mathbf{PD} = \bigcup_\Gamma PD(\Gamma).$$

Thus we obtain the usual notation **1-NPDA** for the family of languages accepted by push-down automata (in contrast to the notation used by Engelfriet et al.).
Analogously we can define the sets **x-NStA**, **x-NNeStA**, **x-NCStA**, and **x-NNstStA** of languages accepted by x-way stack automata, non-erasing stack automata, checking stack automata, and nested stack automata. Moreover, by using **D** instead of **N** and adding **A** we obtain the corresponding families of deterministic and/or augmented stack automata.

Lemma 1. *Let S be a storage type. Then, for any S-automaton A there is an S-automaton B in normal form such that*

$$L(A, S) = L(B, S).$$

Moreover, given A, the construction of B requires only logarithmic space. with respect to the size of A.

Proof. In a first step it is easy to generate an equivalent S-automaton $A' = (Q', X, \delta, q'_{in}, c_{in}, F')$ with $\delta' \subseteq Q' \times X \times BE(P) \times Q' \times \{R, N, L\} \times (I \cup \{\lambda\}$ by the usual adding of new states using the predicate *true*, i.e., a transition $t = (p, x, b, q, r, f_1 f_2)$ is transformed into $t_1 = (p, x, b, p_1^t, N, f_1)$ and $t_2 = (p_1^t, x, q, r, f_2)$.
The main step is now to unfold $BE(P)$-expressions. Any $b \in BE(P)$ may be regarded as a tree which inner nodes are labelled with $\wedge$ or $\vee$ or $\neg$ and the leaves with elements of P. As unique *name* of each node we use the $\{0, 1\}$-description of the path, by which it is branched from the root which is named by λ. Thus the left son of an inner node named v is named by $v0$ and the right son by $v1$.
The idea of the construction of the automaton B is to simulate the tree stepwise. That is, for each transition $t = (p, x, b, q, r, f)$ and each node v in the tree of b we associate new states $q_s^{t,v}$, $q_t^{t,v}$, and $q_f^{t,v}$ (where the lower indices indicate *start, true, false*) and the following transitions:

$$\begin{aligned} t_1 &= (p, x, true, q_s^{t,\lambda}, N, \lambda), \\ t_2 &= (q_t^{t,\lambda}, x, true, q, r, f) \end{aligned}$$

(we start and finish the determination of the truth value of b)

$$
\begin{aligned}
t_1^v &= (q_s^{t,v}, x, true, q_s^{t,v0}, N, \lambda),\\
t_2^v &= (q_t^{t,v0}, x, true, q_t^{t,v}, N, \lambda),\\
t_3^v &= (q_f^{t,v0}, x, true, q_s^{t,v1}, N, \lambda),\\
t_4^v &= (q_t^{t,v1}, x, true, q_t^{t,v}, N, \lambda),\\
t_5^v &= (q_f^{t,v1}, x, true, q_f^{t,v}, N, \lambda)
\end{aligned}
$$

if v is labelled by $\vee$, and analogous transitions if the node is labelled by $\wedge$ or $\neg$, and

$$
\begin{aligned}
t_1^v &= (q_s^{t,v}, x, pr, q_t^{t,v}, N, \lambda),\\
t_2^v &= (q_s^{t,v}, x, \neg pr, q_f^{t,v}, N, \lambda)
\end{aligned}
$$

if the node v is a leaf labelled by the predicate pr. This construction can be done with logarithmic space by using the algorithm of Nancy Lynch (see [13] or [2]). The main idea is to compute the nodename (which may have superlogarithmic length!) of a functionsymbol or of a predicatename in the coding of b by using the position of the input head within b and computing the path (and thus the nodename) iteratively down from the root.

In the sequel we shall assume that P contains also the predicate $true$ and $\neg p$ for any $p \in P$.

Typical results for actual families of storage types are normal forms like the restriction to a binary "working" (e.g. stack) alphabet. Proofs of results of this type consist in stepwise simulations. If we generalize this to abstract storage types we come to the notion of *simulation*. Informally, a storage type S is able to simulate a storage type S' if there is an embedding of the configurations of S' into those of S (e.g. a coding of a tape alphabet into a binary alphabet) such that each instruction of S' may be simulated by a sequence of instructions of S and that for each predicate of S' there exist a sequence of instructions of S, say $g_1 g_2 \ldots g_n$, and a Boolean expression over predicates of S such that the Boolean expression, if the predicates are applied to the configurations obtained when the sequence $g_1 g_2 \ldots g_n$ is executed, gets the same truth value as the given predicate of S'. In addition, we require that the sequence $g_1 g_2 \ldots g_n$ leaves the configuration unchanged, i.e. $g_n \circ \ldots \circ g_1$ is the identity (or noop instruction). (In the case of the simulation of a predicate top_x with a binary alphabet, this would mean that we remove the topmost binary stack elements checking whether they form a binary coding of x, and then we restore the old push-down contents.)

In order to give the formal definition we give the following notation. We set

$$I_n = \{(g_1, g_2, \ldots, g_n) : g_i \in I^*, g_n \circ g_{n-1} \circ \ldots \circ g_1 = id\}$$

For $g \in I^*$ and $(g_1, g_2, \ldots, g_n) \in I_n$, we define the sets

$$P \circ g = \{p \circ g : p \in P\},$$
$$P \circ (g_1, g_2, \ldots, g_n) = \bigcup_{j=0}^{n-1} P \circ (g_j \circ g_{j-1} \circ \ldots \circ g_1).$$

Definition. *i) Let $S = (C, P, I, C_{in}, C_{fin})$ and $S' = (C', P', I', C'_{in}, C'_{fin})$ be two storage types. We say that S simulates S' if there are mappings*

$$\alpha : C' \longrightarrow C,$$
$$\beta : P' \longrightarrow \bigcup_{n \geq 1} \bigcup_{(g_1,\ldots,g_n) \in I_n} BE(P \circ (g_1, \ldots g_n),$$
$$\gamma : I' \longrightarrow I^*$$

such that for all $c \in C'$ and $f \in I'$,

$$\alpha(f(c)) = \gamma(f)(\alpha(c)),$$

for all $b \in P'$ and $c \in C'$,

$$b(c) = true \text{ if and only if } \beta(b)(\alpha(c)) = true,$$

and

$$C_{in} \supseteq \alpha(C'_{in}) \quad \text{and} \quad C_{fin} = \alpha(C'_{fin}).$$

ii) Let $\mathbf{S}$ be a class of storage types. We say that $S_0 \in \mathbf{S}$ is universal for $\mathbf{S}$ if S_0 simulates any $S \in \mathbf{S}$.

Lemma 2. *Let S and S' be two storage types such that S simulates S'. Then for any S'-automaton A, there is an S-automaton B such that*

$$L(A, S') = L(B, S).$$

Proof. Let $S = (C, P, I, C_{in}, C_{fin}), S' = (C', P', I', C'_{in}, C_{fin})$ and the S'-automaton $A = (Q_A, X, \delta_A, q_{in,A}, c_{in,A}, F_A)$ be given, where we assume without loss of generality that A is in normal form. We now construct the S-automaton $B = (Q_B, X, \delta_B, q_{in,B}, c_{in,B}, F_B)$. The definition of Q_B and δ_B can be seen from the following remarks. Let $t = (p, x, b, q, r, f) \in \delta_A$.
In order to determine the Boolean value of $\beta(b)$ on $\alpha(c)$ we use the same method as in the proof of Lemma 1. However, the predicates which we have to apply in the leaves are of the form $\pi \circ g_i \circ g_{i-1} \circ \ldots \circ g_1$, and therefore we use states of the form $(v', g_1 g_2 \ldots g_i)$ where the first component gives the states corresponding to some leaf and the second one can be used to compute the needed subsequence of instructions. The work in the second components is done by transitions of the forms

$$((p', g_1 \ldots g_i), \lambda, true, (p', g_1 \ldots g_j), N, g_{i+1}, g_{i+2} \ldots g_j) \text{ if } i < j,$$
$$((p', g_1 \ldots g_i), \lambda, true, (p', g_1 \ldots g_j), N, g_{i+1} \ldots g_n g_1 \ldots g_j) \text{ if } i > j.$$

Finally we split the application of a sequence of instructions into single instructions as in the proof of Lemma 1. By this construction and $\gamma(f)(\alpha(c)) = \alpha(f(c))$, we obtain

$$(p, w_1\#w_2, c) \vdash_A (q, w_1'\#w_2', f(c))$$

if and only if

$$(p, w_1\#w_2, \alpha(c)) \vdash_B^* (q, w_1'\#w_2', \alpha(f(c)).$$

If we now set $q_{in,B} = q_{in,A}$, $c_{in,B} = \alpha(c_{in,A})$, and $F_B = F_A$, it is easy to see that $L(A, S') = L(B, S)$.

Corollary 3. *Let* $\mathbf{S}$ *be a family of storage types with the universal storage type* S_0.
i) Then, for any $S \in \mathbf{S}$ *and any* S*-automaton* A*, there is an* S_0*-automaton* B *such that*

$$L(A, S) = L(B, S_0).$$

ii) For $x \in \{1, 2\}$,

$$\mathbf{x\text{-}NSA} = \mathbf{x\text{-}N}S_0\mathbf{A}.$$

Analogous equalities hold for the deterministic and/or augmented versions of automata.

3. Emptiness and membership problems

In this section we compare the emptiness problem for one-way automata and the membership problem for two-way automata.
In order to do this we identify the language family with its membership problem. For a storage type S and a class $\mathbf{X}$ of S-automata, we define the emptiness problem as

$$\emptyset\text{-}\mathbf{X}S\mathbf{A} = \{< A >: L(A, S) \neq \emptyset\}.$$

For a family $\mathbf{S}$ of storage types we define $\emptyset\text{-}\mathbf{XSA}$ as the union of the sets $\emptyset\text{-}\mathbf{X}S\mathbf{A}$ taken over $S \in \mathbf{S}$.

Theorem 4. *Let* S *be a storage type. Then*

$$\mathbf{2-NA}S\mathbf{A} = LOG(\emptyset\text{-}\mathbf{1-N}S\mathbf{A}).$$

Proof. Let $S = (C, P, I, C_{in}, C_{fin})$, and let $A = (Q, X, \delta, q_{in}, c_{in}, F)$ be a two-way S-automaton. For a given word x, we set

$$Z = \{w_1\#w_2 : w_1w_2 = x, w_1, w_2 \in X^*\}$$

(Z describes the possible positions of the head on the word x) and

$$B = (Q \times Z, X, \delta', (q_{in}, \#x), c_{in}, F \times Z)$$

where

$$(p, x_1 \dots x_{i-1} \# x_i \dots x_n), \lambda, b, (q, x_1 \dots x_{j-1} \# x_j \dots x_n), N, g) \in \delta'$$

if and only if

$$(p, x_i, b, q, r, g) \in \delta$$

and

$$j = i - 1 \text{ iff } r = L,\ j = i + 1 \text{ iff } r = R, \text{ or } j = i \text{ iff } r = N.$$

This means that the moves of the head and its reading of letters of x in the two-way automaton A are stored in the second component of the state set of B. Therefore $L(B, S) = \emptyset$ or $L(B, S) = \{\lambda\}$ and

$$x \in L(A, S) \text{ if and only if } L(B, S) = \{\lambda\}.$$

By standard arguments (see [9] for an analogous construction) B can be constructed within logarithmic space. Thus

$$L(A, S) \leq_m^L \emptyset\text{–}\mathbf{1} - \mathbf{N}S\mathbf{A}$$

and

$$LOG(\mathbf{2} - \mathbf{N}S\mathbf{A}) \subseteq LOG(\emptyset\text{–}\mathbf{1} - \mathbf{N}S\mathbf{A}).$$

Since $\mathbf{2} - \mathbf{NA}S\mathbf{A} = LOG(\mathbf{2} - \mathbf{N}S\mathbf{A})$ we obtain

$$\mathbf{2} - \mathbf{NA}S\mathbf{A} \subseteq LOG(\emptyset\text{–}\mathbf{1} - \mathbf{N}S\mathbf{A}).$$

In order to prove the converse inclusion we have to show that there is a two-way Sautomaton accepting the descriptions of one-way S-automata accepting a non-empty set. Since the construction of Lemma 1 is a LOG-reduction we can assume that the description gives a one-way automaton in normal form. Then the algorithm given in Figure 1 can be performed by an augmented two-way S-automaton B on the description of $A = (Q, X, \delta, q_{in}, F)$ (note that the storage operations can be performed since the two-way automaton has the same storage type as A).
Obviously B accepts A if and only if the word $w = x_1 x_2 \dots x_n$ obtained as the concatenation of the second components of the chosen elements of δ is accepted by A, i.e. if and only if $L(A, S) \neq \emptyset$.

If we use δ as input alphabet of B and the corresponding transition as input in the definition of δ', then one can easily show by the same method that

BEGIN
$p := q_{in}$
$c := c_{in}$
B: IF $p \in F$ THEN (*accept*, GOTO A) ELSE
choose $(p, x, b, q, r, f) \in \delta$
IF $b(c) = false$ THEN (*reject*, GOTO A) ELSE
$p := q$
$c := f(c)$
GOTO B
A: END

Figure 1

Theorem 4 also holds for deterministic automata. Thus we obtain

Corollary 5. *Let S be a storage type. Then*

$$\mathbf{2-DA}S\mathbf{A} = LOG(\emptyset\mathbf{-1-D}S\mathbf{A}).$$

Corollary 6. *Let S_0 be a universal storage type for the class* **S** *of storage types. Then $\emptyset\mathbf{-1-N}S_0\mathbf{A}$ is* **2-NASA**-*complete.*

Using the known characterizations of the classes of lannguages accepted by different types of augmented stack automata (see [1] ,[9] ,[10]), we obtain

Corollary 7. *i)* $\emptyset\mathbf{-1-NPDA}$ *is* **P**-*complete.*
ii) **∅-1-NStA** *is* **EXPOLYTIME**-*complete.*
iii) **∅-1-NNstStA** *is* **EXPOLYTIME**-*complete.*
iv) **∅-1-NNeStA** *is* **PSPACE**-*complete.*

4. Complete sets

In the last section we saw that, for any abstract storage type the emptiness problem of one-way automata is complete for the corresponding language class of two-way automata. This relationship was shown for many actual storage types before (see [9]). In this section we shall generalize some other theorems as the Chomsky-Schutzenberger theorem and the existence of hardest languages by Greibach. This was already done within the framework of balloon automata and abstract families of automata (see [6]). Here we shall do this within the Engelfriet approach of automata with storage, which allows a surprisingly simple treatment.

Theorem 8. *Let $S = (C, P, I, C_{in}, C_{fin})$ be a storage type. Then there exists a language L_S such that, for any S-automaton A, there are homomorphisms h, g, and a regular set R (all depending on A) with*

$$L(A, S) = h(g^{-1}(L_S) \cap R).$$

Proof. We define $L_S \subseteq (P \cup I \cup \{(,), \#\})^*$ as the set of all words

$$(b_1 \# f_1)(b_2 \# f_2 \ldots (b_n \# f_n)$$

such that $n \geq 0$, $b_i \in P$ and $f_i \in I$ for $1 \leq i \leq n$, and there exist configurations $c_0 \in C_{in}$ and $c_1, c_2, \ldots, c_{n-1} \in C$ and $c_n \in C_{fin}$ with

$$b_i(c_{i-1}) = true \quad \text{and} \quad f_i(c_{i-1}) = c_i$$

for $1 \leq i \leq n$. Now let $A = (Q, X, \delta, q_{in}, c_{in}, F)$ be a one-way S-automaton. By Lemma 1, we can assume that $\delta \subseteq Q \times X \times P \times Q \times \{N, R\} \times (I \cup \{\lambda\})$. Then we define the homomorphism

$$g : (Q \times X \times P \times Q \times \{N, R\} \times (I \cup \{\lambda\}))^* \longrightarrow (P \cup I \cup \{(,), \#\})^*$$

by

$$g((p, x, b, q, r, f)) = (b \# f)$$

and the set $R \subseteq (Q \times X \times P \times Q \times \{R, N\} \times (I \cup \{\lambda\})^*$ as the set of all strings

$$(p_1, x_1, b_1, p_2, r_1, f_1)(p_2, x_2, b_2, p_3, r_2, f_2) \ldots (p_n, x_n, b_n, q_{n+1} r_n, f_n)$$

with $n \geq 0$, $p_1 = q_{in}$, $p_{n+1} \in F$ and $(p_i, x_i, b_i, p_{i+1}, r_i, f_i) \in \delta$ for $1 \leq i \leq n$. Note that R is regular. Then the set $g^{-1}(L_S) \cap R$ consists of all strings

$$w = (p_1, x_1, b_1, p_2, r_1, f_1)(p_2, x_2, b_2, p_3, r_2, f_2) \ldots (p_n, x_n, b_n, p_{n+1}, r_n, f_n) \in R$$

such that there is a sequence $c_0, c_1, \ldots, c_n$ with $c_o = c_{in}$, $f_i(c_{i-1}) = c_i$ and $b_i(c_{i-1}) = true$ for $1 \leq i \leq n$, and $c_n \in C_{fin}$. Thus w resembles an accepting computation. If we define

$$h((p,x,b,q,N,f)) = \lambda \quad \text{and} \quad h((p,x,b,q,R,f)) = x$$

for $p, q \in Q$, $x \in X$, $b \in P$, and $f \in I$, we obtain that $h(w)$ is an element $L(A,S)$. In fact,

$$h(g^{-1}(L_S) \cap R) = L(A,S).$$

If we fix the initial configuration c_0 in the last construction and denote the corresponding set by $L_S(c_0)$, we see that $L(A,S) = h(g^{-1}(L_S(c_0)) \cap R)$ still holds. Moreover, it is not hard to see that $L_S(c_0)$ is in **1-D**S**A**. Hence **1-N**S**A** is the trio generated by **1-D**S**A**. Since **1-D**S**A** is obviously closed under intersection with regular sets and inverse homomorphisms, and **1-N**S**A** is closed under homomorphisms we get

Corollary 9. *For any storage type S, every set in* **1-N**S**A** *is a homomorphic image of a language in* **1-D**S**A**, *or shortly,* $H(\mathbf{1-D}S\mathbf{A}) = \mathbf{1-N}S\mathbf{A}$.

Remark. If the considered one-way (nondeterministic) S-automaton A is *realtime*, i.e. it has no transitions with N-moves of the reading head, then the homomorphism in Theorem 8 and Corollary 9 can be chosen to be nonerasing.

To generalize the Theorem of Greibach to abstract storage types seems to be more difficult. Greibach used in her proof of the result that every context-free language is the inverse homomorphic image of the *hardest context-free language* (see [7]) two important properties of the family of context-free languages:
- each context-free language can be represented by a grammar, which means retranslated to machines, that the involved automaton has just one state and acceptance and rejection is purely controlled by the final configuration, and
- the grammar can be assumed to be in Greibach normal form, which retranslated means that the involved automaton is realtime.
Note that this does not hold for stack languages. Thus, if we try to generalize the Greibach result to some storage type $S = (C, P, I, C_{in}, C_{fin})$ we would come up with some construction like

$$\begin{aligned} V_S(c_0) \;=\; \{&[w_{11}\$...\$w_{1s_1}][w_{21}\$...\$w_{2s_2}]\dots[w_{n1}\$...\$w_{ns_n}] : \\ &n \geq 1, \quad \text{there are } i_1, ..., i_n \text{ such that} \\ &w_{1i_1}w_{2i_2}...w_{ni_n} \in \&L_S(c_0)\} \end{aligned}$$

(compare with [7]). The next construction to get rid of states would be a storage type $S \times A$ to given S-automaton A such that there is an $(S \times A)$-automaton B with one state which satisfies $L(A,S) = L(B, S \times A)$ (although S does not simulate $S \times A$, we still have $\mathbf{x-N}S\mathbf{A} = \mathbf{x-N}(S \times A)\mathbf{A}$!).

The Greibach-like result would then be that, for every *realtime* S-automaton A there is a homomorphism $h - A$ such that

$$L(A,S) = h_A^{-1}(V_{S\times A}(c_0)).$$

This is not the desired result! Therefore we shall now handle this problem within complexity theory.
For context-free languages this was done by Sudborough in [16] ; he showed $LOG(CFL) = \mathbf{NAPDA_{PT}}$ (where the index **PT** indicates an additional polynomial time restriction), and hence $\mathbf{NAPDA_{PT}} = LOG(L_{Gr})$. Thus we shall deal with the following classes: If S is an arbitrary storage type, we denote by $\mathbf{2-NA}S\mathbf{A_{PT}}$ the class of all languages in $\mathbf{2-NA}S\mathbf{A}$, which are acceptable by some augmented S-automaton in polynomial time, that is, for each word in the language, there is an accepting computation the length of which is bounded by some polynomial in the length of the given word. Let $\mathbf{1-NA}S\mathbf{A_{PT}}$ be the corresponding class accepted by one-way automata. Furtheron, let $\mathbf{1-NA}S\mathbf{A_{RT}}$ denote the subclass consisting of languages which are acceptable by one-way augmented S-automata in *realtime*, i.e. there are only R-moves and no N-moves. Finally, we need the class $\mathbf{1-NA}S\mathbf{A_{NRT}}$ of *"nearly realtime"* languages, which are acceptable by automata where the set of transition satisfies

$$\begin{aligned}\delta \subseteq\ & Q\times X\times\Gamma\times P\times Q\times\{R\}\times\Gamma\times\{R,N,L\}\times(I\cup\{\lambda\})\\ & \bigcup\ Q\times X\times\Gamma\times P\times Q\times\{N\}\times\Gamma\times\{R,N,L\}\times\{\lambda\},\end{aligned}$$

where Γ is the alphabet of the working tape, i.e., during steps without move of the input head there is no change of the storage. Then we have

Lemma 10.
a) $\mathbf{1-NA}S\mathbf{A_{RT}} \subseteq \mathbf{1-NA}S\mathbf{A_{NRT}} \subseteq \mathbf{1-NA}S\mathbf{A_{PT}} \subseteq \mathbf{2-NA}S\mathbf{A_{PT}}$,
b) $LOG(\mathbf{1-NA}S\mathbf{A_{RT}}) = LOG(\mathbf{1-NA}S\mathbf{A_{NRT}}) = LOG(\mathbf{1-NA}S\mathbf{A_{PT}})$
$= LOG(\mathbf{2-NA}S\mathbf{A_{PT}}) = \mathbf{2-NA}S\mathbf{A_{PT}}$.

Proof. a) is obvious.
b) $LOG(\mathbf{2-NA}S\mathbf{A_{PT}}) = \mathbf{2-NA}S\mathbf{A_{PT}}$ is done by the well-known construction to show the transitivity of LOG-reducibilities.
$\mathbf{2-NA}S\mathbf{A_{PT}} = LOG(\mathbf{1-NA}S\mathbf{A_{PT}})$ uses the usual repetition construction $w \longrightarrow (w\$)^{p(n)}$ (see [16]), which works since we have polynomial time bounds and the investigated automata are augmented to be able to change from one $\$w\$$-block to the next.
Finally, we indicate a LOG-reducibility from $\mathbf{1-NA}S\mathbf{A_{PT}}$ to $\mathbf{1-NA}S\mathbf{A_{RT}}$. If $A = (Q, X, \Gamma, \delta, q_0, c_{in}, \gamma, F)$ is an augmented one-way S-automaton with polynomial time bound $p(n)$ and $\delta \subseteq Q\times X\times\Gamma\times P\times Q\times\{R,N\}\times\Gamma\times\{L,N,R\}\times(I\cup\{\lambda\})$ (i.e., it is in normal form), we first add the following "no-change"-transitions to δ:

$$(q, x, a, true, q, N, a, N, \lambda) \text{ for } q \in Q,\ x \in X,\ a \in \Gamma.$$

With these additional transitions it is possible to pad each part of N-moves in a computation of exactly $p(n)$ steps. Thus, if v is accepted by A (with no-change-moves), then there is an accepting computation consisting of $|v|$ R-steps each of which is followed by $p(|v|)$ N-steps (not necessarily no-change-moves!). Now we consider the realtime S-automaton B which is obtained from A by converting each N-transition $(q_1, x, a_1, b, q_2, N, a_2, r, f)$ into $(q_1, \#, a_1, b, q_2, N, a_2, r, f)$, where $\#$ is a new input symbol not contained in X. Then the mapping

$$x_1x_2 \dots x_n \longrightarrow f(x_1x_2 \dots x_n) = \#^{p(n)}x_1\#^{p(n)}x_2\#^{p(n)} \dots x_n\#^{p(n)}$$

is computable within logarithmic space and reduces $L(A, S)$ to $L(B, S)$.

Our analogon of Greibachs language is defined as

$$\begin{aligned} V_S(c_0) \;=\; \{&[w_{11}\$...\$w_{1s_1}] \dots [w_{n1}\$...\$w_{ns_n}] : \\ &n \geq 1, w_{ij} \in (\{\S, \#, (,)\} \cup P \cup I)^*, \\ &\text{there are } i_1, ..., i_n \text{ and } m \text{ such that} \\ &w_{1i_1}w_{2i_2}...w_{ni_n} = \S^m(b_1\#f_1)\S^m(b_2\#f_2)\S^m...\S^m(b_n\#f_n)\S^m, \\ &(b_1\#f_1)...(b_n\#f_n) \in L_S(c_0)\} \end{aligned}$$

for any $c_0 \in C_{in}$. Obviously, we get

Lemma 11. $\{V_S(c_0) : c_0 \in C_{in}\} \subseteq \mathbf{1 - NA}S\mathbf{A_{NRT}}$.

Hence $\{V_S(c_0) : c_0 \in C_{in}\} \subseteq \mathbf{2 - NA}S\mathbf{A_{PT}}$.
To show the completeness of this family we use

Theorem 12. $\mathbf{1 - NA}S\mathbf{A_{RT}} \leq_m^L \{V_S(c_0) : c_0 \in C_{in}\}$.

Idea of Proof. Let $A = (Q, X, \Gamma, \delta, q_0, c_{in}, \gamma, F)$ be a realtime augmented S-automaton in normal form, i.e. $\delta = Q \times X \times \Gamma \times P \times Q \times \{R\} \times \Gamma \times \{L, N, R\} \times (I \cup \{\lambda\})$. For a fixed input $v = x_1x_2 \dots x_n \in X^*$ of length n, we consider the set

$$K_n = Q \times \Gamma^{\lceil log(n) \rceil} \times \{0, 1, ..., \lceil log(n) \rceil\}$$

of *surface configurations*, each of which consists of the state, the content of the augmented working tape, the position of the input tape head, and the position of the working tape head. The cardinality $m = \#(K_n)$ is bounded by some polynomial $p(n)$, which is dependent of A only. We order the m surface configurations $K_n = \{k_o, k_1, \dots, k_{m-1}\}$such that k_0 is the initial configuration, i.e. initial state, working tape empty, both heads on the left end

of the tapes. For a transition t in δ it is then possible to say that t *leads* a surface configuration k to another surface configuration k', if the changes of state, working tape and head positions follow the corresponding change induced by t.
Following Greibachs construction (see [7]), for $x \in X$, we define

$$\begin{aligned} H_x &= \{\S^{m-i}(b,f)\S^j : \text{ there is a } (q, x_n, a, b, q', R, a', r, f) \in \delta \\ &\qquad\qquad \text{which leads } k_i \text{ to } k_j\} \\ &= : \{w_1(x), w_2(x), \ldots, w_{s(x)}(x)\}. \end{aligned}$$

Now we set

$$U_i = [w_1(x_i)\$w_2(x_i)\$ \ldots \$w_{s(x_i)}(x_i)]$$

for $2 \leq i < n$ and

$$\begin{aligned} H^{in} &= \{\S^m(b,f)\S^j : \text{ there is } (q_0, x_1, a, b, q, R, a', r, f) \in \delta \\ &\qquad\qquad \text{which leads } k_0 \text{ to } k_j\} \\ &= : \{w_1^{in}, w_2^{in}, \ldots, w_s^{in}\}, \\ U_1 &= \{[w_1^{in}\$w_2^{in}\$ \ldots \$w_s^{in}], \\ H^{fin} &= \{\S^{m-i}(b,f)\S^m : \text{ there is } (q, x, a, b, q', R, a', r, f) \in \delta \\ &\qquad\qquad \text{which leads } k_i \text{ to } k_j \text{ and } q' \in F\} \\ &= : \{w_1^{fin}, w_2^{fin}, \ldots, w_{s'}^{fin}\}, \\ U_n &= [w_1^{fin}, w_2^{fin}, \ldots, w_{s'}^{fin}]. \end{aligned}$$

Now we shall show that the within logarithmic space computable mapping $v \longrightarrow v_1 v_2 ... v_{n-1} v_n = f(v)$ reduces $L(A,S)$ to $V_S(c_0)$. If $f(v)$ is in $V_S(c_0)$ then, by construction, there is a sequence $i_1, i_2, \ldots, i_n$ such that

$$w_{1i_1} w_{2i_2} \ldots w_{ni_n} = \S^m(b_1,f_1)\S^m(b_2,f_2)\S^m \ldots \S^m(b_n,f_n)\&^m$$

with

$$(b_1,f_1)(b_2,f_2)\ldots(b_n,f_n) \in L_S(c_0).$$

Thus there exist indices $j_1, j_2, \ldots, j_{n-1}$ such that

$$\begin{aligned} \S^m(b_1,f_1)\S^{m-j_1} &= w_{1i_1}, \\ \S^{m-j\mu-1}(b_\mu,f_\mu)\S^{j_\mu} &= w_\mu^{i_\mu} \text{ for } 2 \leq \mu \leq n-1, \\ \S^{m-j_{n-1}}(b_n,f_n)\S^m &= w_{ni_n}. \end{aligned}$$

Therefore there is a sequence of surface configurations $k_0, k_{j_1}, \ldots, k_{j_{n-1}}, k_{j_n}$ where $k_{j_{\mu-1}}$ leads to k_{j_μ} by a transition with test b_μ and f_μ. However,

$(b_1, f_1)(b_2, f_2) \dots (b_n, f_n) \in L_S(c_0)$ ensures that the whole string $f(v)$ represents a valid accepting computation of A on v, i.e., $v \in L(A, S)$.
The converse implication that $v \in L(A, S)$ implies $f(v) \in V_S(c_0)$ is straightforward.
Hence we have

$$L(A, S) = f^{-1}(V_S(c_0)).$$

Corollary 13. $\{V_S(c_0) : c_0 \in C_{in}\}$ *is* $\mathbf{2-NA}S\mathbf{A_{PT}}$*-complete.*

The whole construction would be easier if we would restrict to storage types which have just one initial configuration, or equivalently, we would work with codings of configurations (we did so already as we coded S-automata and their initial configurations), and which have certain instructions which allow the installation of configurations according to a coding. We then would deal with languages L_S and V_S. It is easy to see that for all types of stacks considered so far we could work with one initial configuration, and thus we get

Theorem 14. *For* $S \in \{St, NeSt, CSt, NstSt\}$,

$$\mathbf{NP} = \mathbf{2-NA}S\mathbf{A_{PT}} = LOG(V_S).$$

Proof. The statement follows from the **NP**-completeness of checking stack languages and the containment of index languages in **NP**.

5. Discussion and open questions

Throughout this paper we use $LOGSPACE$-reducibilities. But it seems to us that no $LOGSPACE$-complete task had to be performed by any of the reductions constructed in this paper. In most cases, $\mathbf{AC^0}$ or even $DLOGTIME$-reducibilities should be sufficient. The exceptions seem to be Lemma 1 and Lemma 2, where a Boolean formula has to be evaluated and this can be done in $ALOGTIME$ by Buss ([2]). Hence all results in this paper should hold with respect to $\mathbf{NC^1}$-reductions as well.
We stated and proved all results of this paper for nondeterministic automata. Concerning determinism Lemma 10 should also hold as well (compare this with $\mathbf{DAPDA_{PT}} = LOG(\mathbf{DCFL})$ in [16].
A related question would be to exhibit conditions under which $\mathbf{2-NA}S\mathbf{A} = \mathbf{2-DA}S\mathbf{A}$ is true. Except for finite memories and for checking stacks, this relation holds with respect to all storage types known to the authors.
Another question in this context would be to investigate alternation, more generally, the whole setup of automata with storage is a sequential one. What

about circuit-like structures (or $PRAM$s) with storage?
Finally we ask for conditions under which $\mathbf{1-NA}S\mathbf{A_{PT}}$ is reducible to $\mathbf{1-N}S\mathbf{A_{PT}}$. This is, for instance, no problem for push-down, but seems a hard problem for any stack type (without knowledge of the **NP**-completeness of **1-NCStA**). Related (at second sight!) are the problems if **1-N**S**A** is perse polynomially bounded and if **1-N**S**A** has a decidable emptiness problem.

References

[1] C. Beeri, Two-way nested stack automata are equivalent to two-way stack automata. J.Comp.Syst.Sci. 10 (1975) 317-339.

[2] S. Buss, The Boolean formula value problem is in $ALOGTIME$. Proc. 19th STOC, 1987.

[3] S. Cook, Characterizations of pushdown machines in terms of time bounded computers. J.Assoc.Comp.Mach. 18 (1971) 4-18.

[4] J. Engelfriet, Context-free grammars with storages. Techn. Report 86-11, University of Leiden, Department of Computer Science.

[5] J. Engelfriet and H.J. Hoogeboom, Automata with storage on infinite words. LNCS , 289-303.

[6] S. Ginsburg, Algebraic and Automata-Theoretic Properties of Formal Languages. North-Holland, Amsterdam, 1975.

[7] S. Greibach, The hardest context-free language. SIAM J.Comput. 2 (1973) 304-310.

[8] J.Hopcroft and J.Ullman, Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading Mass., 1979.

[9] H.Hunt, On the complexity of finite, pushdown, and stack automata. Math. Systems Theory 10, (1976) 33-52.

[10] O. Ibarra, Characterizations of some tape and time complexity classes of Turing machines in terms of multihead and auxiliary stack automata. J.Comp.System Sci. 5 (1971) 1971.

[11] K.-J. Lange, Complexity theory and formal languages. LNCS 381, 19-36, 1988.

[12] K.-J. Lange and M. Schudy, A further link between formal languages and complexity theory. EATCS Bull. 33 (1987) 67-71.

[13] N. Lynch, $LOGSPACE$ recognition and translation of parenthesis languages. J.Assoc.Comp.Mach. 24 (1977) 583-590.

[14] B. Monien and I.H.Sudborough, The interface between language theory and complexity theory. In R.Book (ed.), Formal Languages - Perspectives and Open Problems, 287-324, Academic Press, 1980.

[15] G. Rozenberg, On a family of acceptors for some classes of developmental languages. Internat.J.Comput.Math. 4 (1974) 218-235.

[16] I.H. Sudborough, On the tape complexity of deterministic context-free languages. J.Assoc.Comp.Mach. 25 (1978) 405-414.

[17] K. Wagner and G. Wechsung, Computational Complexity. Deutscher Verlag der Wissenschaften, Berlin, 1986.