On Distinguishing $\mathbf{NC^1}$ and NL

Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig

WSI - University of Tübingen, Germany, Sand 13, 72076 Tübingen, Germany.
{krebs, lange, ludwigm}@informatik.uni-tuebingen.de

Abstract. We obtain results within the area of dense completeness, which describes a close relation between families of formal languages and complexity classes. Previously we were able show that this relation exists between counter languages and **NL** but not between the regular languages and $\mathbf{NC^1}$.
We narrow the gap between the regular languages and the counter languages by considering visibly counter languages. It turns out that they are not densely complete for $\mathbf{NC^1}$. At the same time we found a restricted counter automaton model which is densely complete for **NL**.
Besides counter automata we show more positive examples in terms of L-systems.

1 Introduction

Turing machines are the key model for computation and the most general as well. A consequence however is for example the undecidability of the word problem. Two of the major areas of theory can be understood as different branches originating from the concept of the Turing machine. One branch limits Turing machines in terms of resources like space and time which led to what we know as complexity theory, the study of complexity classes. In the other branch we are in a way limiting the functionality of Turing machines resulting e.g. in pushdown or finite automata. We want to name the objects of the second branch *families of formal languages*. It turned out that complexity classes and families of formal languages have very different properties but they are also connected in many ways. The present work is a contribution to understanding the relationship between complexity classes and families of formal languages. We hope that this leads to new insights to complexity as it is much harder to analyze compared to families of formal languages.

The term *complexity class* is clear, the term *family of formal languages* however needs clarification. Certainly one can interpret *formal language* in a way, that every subset of Σ^* is a formal language but we understand it, as outlined above, as languages which are accepted or generated by certain objects like automata or grammars. Finding a final definition of what we want to consider a family of formal languages is part of our ongoing work.

The regular languages represent a very basic example of a large abundance of families of formal languages, coined by pumping theorems and built on that decision properties, which distinguish them from complexity classes. Nevertheless, most of them exhibit very close relationships to complexity classes.

General algorithms in terms of Turing machines or circuit families are immune to a combinatorial or algebraic analysis. This makes families of formal languages interesting as they contain problems complete for complexity classes and thus in their word problems exhibit close connections to complexity theory. At the same time they are restricted in a way which makes them open for a combinatorial and even algebraic analysis.

It indeed is often the case that a family of formal languages $\mathcal{F}$ is complete for a complexity class $\mathcal{C}$ in the sense that $\mathcal{F}$ is contained in $\mathcal{C}$, and $\mathcal{F}$ contains a $\mathcal{C}$-complete problem. Examples for this situation abound in circuit complexity, e.g., with the regular languages and $\mathbf{NC^1}$, or the context-free languages and $\mathbf{SAC^1}$. Strengthening this link, the notion of dense completeness [KL12] further requires that each $C \in \mathcal{C}$ corresponds to a formal language $F \in \mathcal{F}$ of the same complexity, i.e. such that C and F are reducible to each other.

While it is usual to have a complete family of formal languages corresponding to some complexity class, the picture is different for dense completeness. Up to now we only found dense completeness in non-deterministic classes. Also the proofs heavily rely on non-determinism. Our working hypothesis is that only non-deterministic classes have a densely complete family of formal languages. The first examples established in [KL12] are the following:

- The index languages are densely complete for $\mathbf{NP}$.
- The context-free languages are densely complete for $\mathbf{SAC^1}$.
- The nondeterministic one-counter languages are densely complete for $\mathbf{NL}$.
- The regular languages are **not** densely complete for $\mathbf{NC^1}$.

We are especially interested in non-denseness results. In the instance of the regular languages the proof relies on the gap result of [BCST92]. In [KLL15] we were able to derive a corresponding gap result for the family of visibly counter languages, which we will use in this paper to show unconditionally that even the visibly counter languages are not densely complete in $\mathbf{NC^1}$.

Our results lead to an interesting situation: We have a counter-based family which is densely complete for $\mathbf{NL}$ and one which is not densely complete for $\mathbf{NC^1}$. The next logical step of course would be to find out whether the deterministic one-counter languages are densely complete for $\mathbf{L}$.

A different direction is to look at the visibly pushdown languages which are also $\mathbf{NC^1}$ complete [Dym88]. We cannot rule out the possibility that $\mathbf{NC^1}$ contains indeed a dense family of formal languages. However it seems easier to show non-denseness for $\mathbf{NC^1}$. On the other hand it seems easy to show denseness results for non-deterministic classes. We present examples of families of formal languages which are densely complete for certain complexity classes in terms of L-systems. This underlines our assumption that dense completeness captures an inherent and important property.

Due to space restriction, we omit some of the proofs.

We thank the anonymous referees.

2 Families of Formal Languages

In this section we recollect some notions and results of classes which we will call *families of formal languages*. They have in commen that the complexities of their word problems typically range between $\mathbf{AC^0}$ and $\mathbf{NP}$. In contrast to complexity classes they exhibit pumping or iteration properties which lead to the decidability of emptiness and finiteness of their members, but typically not to that of equivalence or universality.

At present, we have no finished definition of a family $\mathcal{F}$ to be a family of formal languages. What we assume as minimum requirements are:

Recursive presentability: There is recursively enumerable set $R \subseteq \Sigma^*$ and a mapping ϕ from R into the powerset of Σ^*. Each $x \in R$ represents a language generating device, e.g.: a grammar or an automaton, which generates the language $\phi(x)$ and we have $\mathcal{F} = \{\phi(x) | x \in R\}$.

Decidabilities: The emptiness and the finiteness of elements of $\mathcal{F}$, i.e. the sets $\{x \in R | \phi(x) \neq \emptyset\}$ and $\{x \in R | \phi(x) \text{ is finite}\}$, are decidable.

Closure properties: $\mathcal{F}$ is constructively closed under intersection with regular sets and under inverse morphisms. These closures are constructively in the sense, that from $x \in R$, morphism h, or given finite automaton A a $y \in R$ is computable, such that $\phi(y) = \phi(x) \cap L(A)$ resp. $h^{-1}(\phi(x))$.

Unfortunately, we can construct language families, which we do not regard as family of formal languages, but which in fact fulfill these properties. Thus we go by well-known examples like the family of regular set, that of context-free languages and some of their various subfamilies, and some families of context-free Lindenmayer languages, which all fulfill the requirements mentioned above.

Notation. We fix a finite alphabet Σ. By Σ^* we denote the set of words over Σ. A language over Σ is a subset of Σ^*. For $w \in L \subseteq \Sigma^*$, by $|w|$ we denote the length of w and ϵ is the word of length 0. For $A \subseteq \Sigma$ we denote by $|w|_A$ the number positions in w having a letter in A. For a word, w_i is the letter in position i.

Regular languages. The regular languages are a prime example for a family of formal languages and indeed fulfill all the requirements we proposed for something being called a family of formal languages. They can be defined in terms of finite automata (DFA and NFA), logic (MSO), and algebra (finite syntactic monoid).

Context-free languages. Context-free languages (CFL) correspond to those accepted by pushdown automata (PDA). The deterministic variant is strictly weaker. The same goes for counter languages. Here only one stack symbol may be pushed by the accepting automaton. We write NOCA for non-deterministic one-counter automata and DOCA for the deterministic variant.

Visibly pushdown languages.

Context-free languages are accepted by pushdown automata. A way to restrict pushdown automata has received much attention in the last ten years. The visibility restriction for pushdown automata leads to the class of visibly pushdown languages (a.k.a. input-driven pushdown languages), short: VPL. Here, the input symbol determines the stack operation, i.e. if a symbol is pushed or popped. This leads to a partition of Σ into call, return and internal letters: $\Sigma = \Sigma_{\text{call}} \cup \Sigma_{\text{ret}} \cup \Sigma_{\text{int}}$. Then $\hat{\Sigma} = (\Sigma_{\text{call}}, \Sigma_{\text{ret}}, \Sigma_{\text{int}})$ is a visibly alphabet. In the rest of the paper we always assume that there is a visibly alphabet for Σ if we speak about VPL.

We define a function $\Delta : \Sigma^* \to \mathbb{Z}$ which gives us the *height* of a word by $\Delta(w) = |w|_{\Sigma_{\text{call}}} - |w|_{\Sigma_{\text{ret}}}$. Each word w over a visibly alphabet can be assigned its *height profile* w^{Δ}, which is a map $\{0, \ldots, |w|\} \to \mathbb{Z}$ with $w^{\Delta}(i) = \Delta(w_1 \cdots w_i)$. Mehlhorn [Meh80] and independently also Alur and Madhusudan [AM04] introduced *input-driven* or *visibly pushdown automata* (VPDA). In these automata the input letter determines the kind of stack operation. We omit a formal definition for VPDA here, as we are actually interested in a more restricted model.

The family of languages which are accepted by some VPDA is called VPL. This family enjoys many constructive closure and decision properties.

In [BLS06], a reasonable restriction of VPDA was introduced by visibly counter automata (VCA). That is a counter automaton which obeys the visibly restriction. In [BLS06] this model was used as a tool for showing a certain problem concerning VPL to be decidable. In particular they showed that given a VPDA, it is decidable if the language is accepted by some VCA. The following definition exhibits a natural m which we will call threshold. It allows the automaton to have a limited access to the current stack height.

Definition 1 (m-VCA). *An m-VCA $\mathcal{A}$ over $\hat{\Sigma} = (\Sigma_{\text{call}}, \Sigma_{\text{ret}}, \Sigma_{\text{int}})$ is a tuple $\mathcal{A} = (Q, q_0, F, \hat{\Sigma}, \delta_0, \ldots, \delta_m)$, where $m \geq 0$ is the threshold, Q is the set of states, q_0 the initial state, F the set of final states, and $\delta_i \colon Q \times \Sigma \to Q$ are the transition functions.*

A configuration is an element of $Q \times \mathbb{N}$. Note that m-VCAs, similar to VPDAs, can only recognize words where the height profile is non-negative. All other words are rejected. An m-VCA $\mathcal{A}$ performs the following transition when a letter $\sigma \in \Sigma$ is read: $(q, k) \xrightarrow{\sigma} (\delta_{\min(m,k)}(q, \sigma), k + \Delta(\sigma))$. Then $w \in L(\mathcal{A})$ iff $(q_0, 0) \xrightarrow{w} (f, \delta(w))$ for $f \in F$.

The class of the visibly counter languages (VCL) contains the languages recognized by an m-VCA for some m.

As previously argued, VPL has many nice properties and is still expressible enough for many applications. The class VCL is even simpler and can function as an intermediate step if we want to extend results form the regular domain to, say, VPL.

Lindenmayer systems. The models we looked at so far are automata-based. Lindenmayer introduced a formal rewriting system similar to grammars whose purpose was to model growth of plants. The main difference is that each leaf in the derivation tree of L-Systems has to have the same depth in contrast to ordinary grammars. One can see the resulting objects as fractals. Besides describing biological processes, L-systems have gotten applied in other fields like computer graphics. L-systems have also found their way in the theory of formal languages. Refer e.g. Rozenberg and Salomaa [RS80].

We call a map $h \colon \Sigma \to 2^{\Sigma^*}$ a *substitution* if $h(\epsilon) = \epsilon$ and $h(uv) = h(u)h(v)$.

Definition 2. *The following are L-systems:*

- *An 0L system is a tuple $G = (\Sigma, h, w)$ where Σ is the alphabet, $h\colon \Sigma \to 2^{\Sigma^*}$ a substitution and $w \in \Sigma^*$ is a word we call axiom. The language of G is $L(G) = \bigcup_k h^k(w)$.*
- *An E0L system is a tuple $G = (\Sigma, h, w, \Delta)$ where (Σ, h, w) is a 0L system and $\Delta \subseteq \Sigma$ is a set of terminals. The language of G is $L(G) = \bigcup_k h^k(w) \cap \Delta^*$.*
- *An ET0L system is a tuple $G = (\Sigma, H, w, \Delta)$, where H is a finite set of substitutions and for every $h \in H$, (Σ, h, w) is a 0L system. The language of G is $L(G) = \{x\Delta^* \mid \exists k \in \mathbb{N} \exists h_1, \ldots, h_k \in H\colon x \in h_1(h_2(\ldots h_k(w)\ldots))\}$.*
- *An ET0L system $G = (\Sigma, H, w, \Delta)$ is an EDT0L system if for all $h \in H$ and for all $a \in \Sigma$ holds that $|h(a)| = 1$, i.e. h is an endomorphism.*
- *An EDT0L system $G = (\Sigma, H, w, \Delta)$ is an ED0L system, if $|H| = 1$.*

3 Complexity Classes

In complexity theory we are interested in the amount of resources needed for solving the word problem. Turing machines are the standard model resulting in the resource measures time, space and (non-)determinism.

Using the logarithmic space bound, we get the nondeterministic class **NL** and using polynomial time bound, we get **NP**.

Circuits. When considering very low complexity classes, other models of computations are needed. A circuit is a directed acyclic graph where the nodes are labeled with Boolean functions unless it is an input node. A word over $\{0, 1\}$ is accepted by the circuit if the output gate results to 1, whereas the result is computed in the obvious way. We only consider $\{0, 1\}$ as an input alphabet; other alphabets can be simulated. If we want to accept languages we naturally want to accept words ob arbitrary length. To achieve this we speak of families of circuits $(C_n)_{n\in\mathbb{N}}$. Here there is one circuit of each input length. If there is some resource-bounded machine computing $(C_n)_{n\in\mathbb{N}}$, we speak of uniformity. If we do not require such a machine, we say, that the circuit family is non-uniform. If not stated otherwise, circuit families are assumed to be non-uniform

Typical complexity measures in circuits are size, depth, fan-in of the gates, type of the gates and uniformity. We get for example the following classes:

- $\mathbf{AC}^i$: circuits of polynomial size, depth in $\mathcal{O}(\log^i(n))$, unbounded fan-in and Boolean gates.
- $\mathbf{ACC}^i$: circuits of polynomial size, depth in $\mathcal{O}(\log^i(n))$, unbounded fan-in and Boolean and modulo gates. If we want to emphasize the modulus k, we denote this by using the notation $\mathbf{ACC}^i_k$.
- $\mathbf{TC}^i$: circuits of polynomial size, depth in $\mathcal{O}(\log^i(n))$, unbounded fan-in and threshold gates.
- $\mathbf{NC}^i$: circuits of polynomial size, depth in $\mathcal{O}(\log^i(n))$, bounded fan-in and Boolean gates.
- $\mathbf{SAC}^i$: circuits of polynomial size, depth in $\mathcal{O}(\log^i(n))$, semi-unbounded fan-in and Boolean gates.

In particular we are interested in the following classes: $\mathbf{AC^0} \subset \mathbf{ACC}^0 \subseteq \mathbf{TC}^0 \subseteq \mathbf{NC^1} \subseteq \mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{SAC^1} \subseteq \mathbf{P}$. Note that the classes $\mathbf{AC^0}$ and $\mathbf{ACC}^0$ are seperated. All other inclusions are unknown whether they are strict. Refer e.g. to [Vol99].

Complexities of families of formal languages. The various families mentioned so far, have the following connections to complexity classes in terms of completeness results. When we say that a family $\mathcal{F}$ is complete for $\mathcal{C}$, we mean that both $\mathcal{F} \subset \mathcal{C}$ and that $\mathcal{F}$ contains a $\mathcal{C}$-complete problem.

- The $ETOL$-languages are $\mathbf{NP}$-complete.
- Both the context-free languages and the EOL-languages are $\mathbf{SAC^1}$-complete.
- Both the nondeterministic counter languages and the $EDTOL$-languages are $\mathbf{NL}$-complete.
- the regular languages, the visibly counter languages, and the visibly push-down languages are $\mathbf{NC^1}$-complete.
- The $EDOL$-languages are $\mathbf{AC^0}$-complete (which is rather the case because of the $\mathbf{AC^0}$ reductions we chose).

Dense completeness. Finally we state the definition of dense completeness as it is introduced in [KL12]. In our setting we use many-one-reductions. If a language A is reducible to B we write $A \leq B$. If $A \leq B$ and $B \leq A$ we write $A \approx_m B$ and say A and B are many-one-equivalent. The reductions we use are all DLOGTIME-uniform $\mathbf{AC^0}$ reductions, so we write e.g. $A \approx_m^{\mathbf{AC^0}} B$.

Definition 3. *Let $\mathcal{F}$ and $\mathcal{C}$ be sets of languages. We say $\mathcal{F}$ is densely complete in $\mathcal{C}$ if*

- *$\mathcal{F} \subseteq \mathcal{C}$ and*
- *for all $C \in \mathcal{C}$ there exists a language $F \in \mathcal{F}$ such that $C \approx_m^{\mathbf{AC}^0} F$.*

The gist of the definition is that we can say that a family of formal languages is densely complete in some complexity class. However in the definition we do not require $\mathcal{F}$ and $\mathcal{C}$ to be restricted in any way. Like that we get transitivity of the dense completeness property which is desirable.

4 Negative Instances for $\mathbf{NC^1}$

As all the examples we know, where a densely complete family of formal languages exists correspond to a (more or less) non-deterministic complexity class, it is rather interesting to consider deterministic classes. For its closeness to $\mathbf{L}$ we consider $\mathbf{NC^1}$ as a deterministic class, and hence would like to show that there is no densely complete family of formal languages in it.

Our approach to prove that a family of formal languages is not densely complete will show that the class is too sparse inside of $\mathbf{NC^1}$. We have shown this before for the regular languages using finite monoids, and it was not clear this approach would ever work for any family of formal languages outside the regular languages. Using a result from a recent paper [KLL15] we are now able to break this barrier and show a family of formal languages outside the regular language that is not densely complete in $\mathbf{NC^1}$. In this section we will show that VCL is not densely complete in $\mathbf{NC^1}$. This also narrows the gap between the counter languages which are densely complete in a complexity class and the regular languages which are not densely complete in a complexity class to counter languages vs. visibly counter languages.

Theorem 4. *The visibly counter languages are not densely complete in* $\mathbf{NC^1}$.

Proof. We show the statement by contradiction. The contradiction will be achieved by using two facts:

- Ladner's theorem in the generalized version by Vollmer [Vol90] shows us how to get arbitrarily long lists of languages L_i for which $L_i \leq L_j$ iff $i \leq j$. That means that complexity classes in a way have infinitely many ascending levels of complexity inside.
- Visibly counter languages (and regular languages as well) exhibit a kind of dichotomy [KLL15] when it comes to the membership of a language to $\mathbf{AC^0}$. Either a visibly counter language is in $\mathbf{AC}^0$ or it is hard for a proper superclass of $\mathbf{AC^0}$.

So it seems likely that those to facts contradict each other and this is what we will prove.

Assume VCL to be densely complete in $\mathbf{NC^1}$. Then for all languages L in $\mathbf{NC^1}$ there exists a language V in VCL such that L and V are many-one-equivalent under $\mathbf{AC^0}$-reductions.

The language PARITY$= \{w \in \{0,1\} \mid |w| \equiv 0 \pmod 2\}$ is not in $\mathbf{AC^0}$ but $\mathbf{ACC}^0_2$-complete [FSS84]. Using Ladner's theorem we choose L to be a language whose complexity lies strictly between $\mathbf{AC^0}$ and $\mathbf{ACC}^0_2$, i.e. $L \leq$ PARITY, PARITY $\not\leq L$ and $L \notin \mathbf{AC^0}$ [Hås86]. The construction for L basically takes a subset of PARITY by only allowing certain word lengths.

Having L we look at V, which must be many-one-equivalent to L:

- If V is in $\mathbf{AC^0}$ then we have a contradiction, since L is not in $\mathbf{AC^0}$.
- If V is not in $\mathbf{AC^0}$ then according to [KLL15], we have to consider again two cases for the two different reasons a visibly counter languages can be outside $\mathbf{AC^0}$. One reason is that the height behavior is too complex, resulting in V being $\mathbf{TC}^0$-hard. The other reason concerns the regular part of the language and is related to the case for regular languages, resulting in V being $\mathbf{ACC}^0_k$-hard for some k. So in both cases, V is hard for a proper superclass.
  - V is hard for $\mathbf{TC}^0$. This is a contradiction, since L is not $\mathbf{ACC}^0_2$-hard.
  - V is hard for $\mathbf{ACC}^0_k$-hard. If k is even then again we have the contradiction because L is not $\mathbf{ACC}^0_2$-hard. If k is odd then this implied $\mathbf{ACC}^0_k \subseteq \mathbf{ACC}^0_2$. Due to [Smo87] we know that this is contradictory also. □

Since the regular languages are visibly counter languages and $\mathbf{NC^1}$-complete, we get the following.

Corollary 5. *The regular languages are not densely complete in* $\mathbf{NC^1}$.

This completes a proof from the previous paper on dense completeness [KL12]. The statement for the non-denseness of the regular languages was true, but the proof was incomplete by not considering the case that the syntactic monoid of a regular language in AC^0 might contain in fact a nontrivial group.

As we saw in the proof, we used Ladner's theorem on the one hand and some kind of dichotomy on the other. Up to now we do not know any other proof strategy for showing that a formal language class is not densely complete in some complexity class.

5 Positive Instances for NL

We introduce a restricted counter-based automaton model being densely complete in **NL**. It can then be used to demonstrate that a certain type of L-system is also densely complete in **NL**.

The automaton model we introduce is a non-deterministic counter automaton with the restriction that once the automaton performs a pop action on the stack, it has to pop until it is empty. See figure 1. We call it a sweeping counter automaton (SCA) and the corresponding family of formal languages we call Scl.

Definition 6. *A nondeterministic sweeping counter automaton (SCA) is a tuple* $\mathcal{A} = (Q^{\uparrow}, Q^{\downarrow}, \Sigma, q_0, F, \delta, \delta_0)$*, where* $Q = Q^{\uparrow} \dot{\cup} Q^{\downarrow}$ *and* $Q^{\uparrow}$ *is a set of push-states,* $Q^{\downarrow}$ *a set of pop-states,* $q_0 \in Q^{\uparrow}$ *is the initial state,* $F \subseteq Q^{\downarrow}$ *a set of final states and* $\delta \subseteq (Q^{\uparrow} \times \Sigma \times Q) \cup (Q^{\downarrow} \times \Sigma \times Q^{\downarrow})$ *and* $\delta_0 \subseteq Q^{\downarrow} \times \Sigma \times Q$ *are transition functions. The transition* δ_0 *is applied if the counter is* 0 *and* δ *is applied otherwise.*

A configuration of a SCA $\mathcal{A}$ is an element of $Q \times \mathbb{N}$. The transition relations δ and δ_0 take an input word and define a run through configurations. The initial configuration is $(q_0, 0)$. Further if $q \in Q^{\uparrow}$ and $k > 0$ then $(q,k) \overset{a \in \Sigma}{\to} (\delta(q,a), k+1)$ and $(q,k) \overset{a \in \Sigma}{\to} (\delta(q,a), k-1)$ in the case of $q \in Q^{\downarrow}$. If $k = 0$ then $(q,k) \overset{a \in \Sigma}{\to} (\delta_0(q,a), k')$ where k'=0 iff $\delta_0(q,a) \in Q^{\downarrow}$; otherwise $k' = 1$. Then: $L(\mathcal{A}) = \{w \in \Sigma^* \mid (q_0, 0) \overset{w}{\to} (f, 0), f \in F\}$. Note that there are some similar ways to define a SCA but the present definition serves our purpose.

Scl is closed under union, intersection, Kleene star and inverse homomorphisms but not under complement. The regular languages are contained in Scl. Further the decidabilities of NOCA translate to Scl. SCA cannot be determinized.

Theorem 7. Scl *is densely complete for* **NL**.

We can use this result to prove denseness of the L-system EDT0L. It is sufficient to show that a densely complete family is a subset of EDT0L and that EDT0L lies within **NL**. The latter is known to be true [RS80].

Lemma 8. Scl $\subseteq$ *EDT0L*

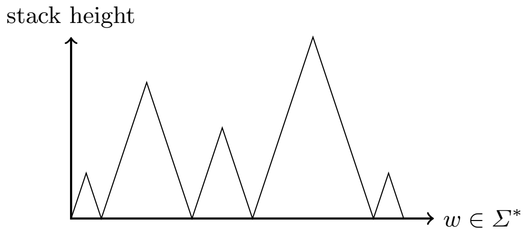

Fig. 1. Characterizing stack height behavior for an SCA.

Proof. We are given an SCA $\mathcal{A} = (Q^{\uparrow}, Q^{\downarrow}, \Sigma, q_0, F, \delta, \delta_0)$ and construct an EDT0L system $G = (\Sigma_G, H, w, \Delta)$ with $\Delta = \Sigma$. For convenience we assume $\epsilon \notin L(\mathcal{A})$. We set $\Sigma_G = \{(q_1, q_2) \mid q_1, q_2 \in Q^{\downarrow}\} \cup \{(q, *) \mid q \in Q^{\downarrow}\} \cup \Delta$.

The idea is to let G generate a letter for each push-pop-cycle of the SCA and then extend each of those letters to the actual word the automaton reads. Hence we set $w = (q_0, *)$. We need a first set of substitutions for extending one more push-pop-cycle. Hence

$$h_1^q((q', *)) = (q', q)(q, *)$$

for all $q' \in Q^{\downarrow}$. In general for all substitutions: On letters not specified the map is the identity. For the final state we need the following variant:

$$h'^q_1((q', *)) = (q', q)$$

for all $q \in F$. Next we have to build the word as the counter in- and decreases.

$$h_2^{a,b}((q, q')) = a(q'', q''')b$$

if $q \in Q$, $q', q''' \in Q^{\downarrow}$, $q'' \in Q^{\uparrow}$ and $q' \in \delta(q''', b)$. Further if $q \in Q^{\downarrow}$ it must hold that $q'' \in \delta_0(q, a)$ and $q'' \in \delta(q, a)$ else. Finally if the final stack height is reached we have to terminate: $h_3((q, q)) = \epsilon$. Now H is the set of all substitutions we just described.

We verify that the construction is correct. Each word $w = w_{(1)} \ldots w_{(k)}$ is accepted by $\mathcal{A}$ where every $w_{(i)}$ corresponds to one push-pop-cycle of the automaton. If $\mathcal{A}$ is in q after $w_{(1)}$ then we use the derivation $(q_0, *) \to (q_0, q)(q, *)$. If $w_{(1)} = aw'b$, then we can apply $h_2^{a,b}$ and so on. It is easy to see that we can derivate as such: $(q_0, *) \to (q_0, q)(q, *) \to \cdots \to w_{(1)}(q, *)$. After that we can proceed with $w_{(2)}$ etc. until the whole word is derivated. Conversely every word we get by the grammar is also a word accepted by the automaton. The only thing we have to note is that if we derivate like this $(q_0, *) \to \cdots \to (q_0, q)(q, q')(q', q'')(q'', *)$ instead of building the word for each cycle first, we just get additional ways to derivate words. $\square$

In conclusion we get the result:

Theorem 9. *EDT0L is densely complete for* **NL**.

6 More densely complete L-systems

An easy case is ED0L, which is densely complete for $\mathbf{AC^0}$ because of the reductions. Further since CFL $\subseteq$ E0L $\subseteq$ $\mathbf{SAC^1}$, we get that E0L is densely complete for $\mathbf{SAC^1}$.

In the previous section we showed that EDT0L is densely complete for **NL**. It is natural to ask whether e.g. the **NP**-complete L-system ET0L is in fact densely complete for **NP**. Using the so-called checking-stack pushdown automata characterization of ET0L [vL76], we can translate proofs from [KL12] to this case. There we showed that PDA are densely complete for $\mathbf{SAC^1}$.

Checking-stack pushdown automata (CS-PDA) are a well researched model [vL76,RS80] in the context of L-systems. A CS-PDA is basically a PDA equipped with a checking stack. This additional stack is nondeterministically filled with some word and after that the checking stack is read only. Further the head for the checking stack is synchronized with the normal stack. This enables the automaton to perform tasks, a usual pushdown automaton cannot do. The checking stack can be used for synchronization of different parts of the computation. E.g. the language $\{ww \mid w \in \Sigma^*\}$ is an easy example which is accepted by an CS-PDA, but by no PDA.

For PDA we have the two-way-variant 2-PDA and also we can use k input heads, which is denoted by $2 - \mathrm{PDA}(k)$. We always restrict such automata to polynomial time and write $2 - \mathrm{PDA}(k)_{\text{poly-time}}$. Using the same notation we get the corresponding CS-PDA models.

Theorem 10 ([vL76]). *ET0L equals CS-PDA and is* **NP***-complete.*

Also by [vL75] it is easy to see that:

Lemma 11. *2-CS-PDA(k) equals* **NP***.*

The proof strategy is very similar to the case of PDA and $\mathbf{SAC^1}$. We first show that going from two-way to one-way preserves denseness and then that the number of heads can be reduced without losing denseness. In conclusion we get:

Theorem 12. *ET0L is densely complete for* **NP***.*

7 Discussion

In this paper we primarily inspected aspects of the relationship between $\mathbf{NC^1}$ and **NL**, using the notion of dense completeness. We developed further the theory of dense completeness and gathered evidence that strengthens our confidence that dense completeness capturs an essential property of nondeterministic complexity.

Dense completeness might offer new angles to separate complexity classes. For instance if $\mathbf{NC^1}$ has no densely complete family of formal languages, we know the class to be strictly contained in **NL**. We could also try to show that every complete family of formal languages in **NL** is already densely complete.

We conjecture that dense completeness is a feature of nondeterministic classes. Of course this should be hard to show since such a result would separate determinism from nondeterminism. To obtain such arguments we need a formal definition of what a family of formal languages should be. In the present work we exhibited one possibility, which is our first contribution.

Our second contribution is showing concrete denseness and non-denseness results. We want to narrow the gap between $\mathbf{NC^1}$ and $\mathbf{NL}$. Hence we want to find large families in $\mathbf{NC^1}$ and show that they are not densely complete. In $\mathbf{NL}$ we want to achieve the opposite and find small families being densely complete. For the $\mathbf{NC^1}$ case we performed this with VCL. In the $\mathbf{NL}$ case we proposed (to our best knowledge) new variant of counter languages, namely sweeping counter languages. As a byproduct we could show the L-system of EDT0L to be densely complete in $\mathbf{NL}$ and ET0L to be densely complete in $\mathbf{NP}$. Also E0L is densely complete for $\mathbf{SAC^1}$ and E0L for $\mathbf{AC^0}$.

There are some interesting questions arising from this work which we will pursue. One of them: Can we show that the deterministic counter languages are not densely complete in $\mathbf{L}$? We are also interested in showing non-denseness of the visibly pushdown languages for $\mathbf{NC^1}$. This would relate rather to the context-free languages which are densely complete for $\mathbf{SAC^1}$.

The only way we know to show non-denseness is using some kind of *dichotomy* of the family of formal languages regarding $\mathbf{AC^0}$. The regular languages for instance are either in $\mathbf{AC^0}$ or hard for $\mathbf{ACC}_0^k$. Not having this dichotomy is actually a weaker property than dense completeness. It is worth investigating how denseness is related to this dichotomy.

The dichotomy for VCL relies on two properties: The regular part must be quasi-aperiodic and the height behavior of the language must be simple. The second one is interesting if we compare this to our new found SCA. In the visibly case, the property of being sweeping would be sufficient for the language to have simple stack behavior. So in the case of visibly sweeping automata, only the regular part determines whether the language is in $\mathbf{AC^0}$. Hence we will continue investigating height behavior properties of counter languages.

References

[AM04] Rajeev Alur and P. Madhusudan. Visibly pushdown languages. In László Babai, editor, *Proceedings of the 36th Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, June 13-16, 2004*, pages 202–211. ACM, 2004.

[BCST92] David A. Mix Barrington, Kevin J. Compton, Howard Straubing, and Denis Thérien. Regular Languages in NC^1. *J. Comput. Syst. Sci.*, 44(3):478–499, 1992.

[BLS06] Vince Bárány, Christof Löding, and Olivier Serre. Regularity Problems for Visibly Pushdown Languages. In Bruno Durand and Wolfgang Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23-25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 420–431. Springer, 2006.

[Dym88] Patrick W. Dymond. Input-Driven Languages are in log n Depth. *Inf. Process. Lett.*, 26(5):247–250, 1988.

[FSS84] Merrick L. Furst, James B. Saxe, and Michael Sipser. Parity, Circuits, and the Polynomial-Time Hierarchy. *Mathematical Systems Theory*, 17(1):13–27, 1984.

[Hås86] Johan Håstad. Almost Optimal Lower Bounds for Small Depth Circuits. In Juris Hartmanis, editor, *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, pages 6–20. ACM, 1986.

[KL12] Andreas Krebs and Klaus-Jörn Lange. Dense Completeness. In Hsu-Chun Yen and Oscar H. Ibarra, editors, *Developments in Language Theory - 16th International Conference, DLT 2012, Taipei, Taiwan, August 14-17, 2012. Proceedings*, volume 7410 of *Lecture Notes in Computer Science*, pages 178–189. Springer, 2012.

[KLL15] Andreas Krebs, Klaus-Jörn Lange, and Michael Ludwig. Visibly counter languages and constant depth circuits. In Ernst W. Mayr and Nicolas Ollinger, editors, *32nd International Symposium on Theoretical Aspects of Computer Science, STACS 2015, March 4-7, 2015, Garching, Germany*, volume 30 of *LIPIcs*, pages 594–607. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[Meh80] Kurt Mehlhorn. Pebbling Moutain Ranges and its Application of DCFL-Recognition. In J. W. de Bakker and Jan van Leeuwen, editors, *Automata, Languages and Programming, 7th Colloquium, Noordweijkerhout, The Netherland, July 14-18, 1980, Proceedings*, volume 85 of *Lecture Notes in Computer Science*, pages 422–435. Springer, 1980.

[RS80] Grzegorz Rozenberg and Arto Salomaa. *Mathematical Theory of L Systems.* Academic Press, Inc., Orlando, FL, USA, 1980.

[Smo87] Roman Smolensky. Algebraic Methods in the Theory of Lower Bounds for Boolean Circuit Complexity. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 77–82. ACM, 1987.

[vL75] Jan van Leeuwen. The Membership Question for ET0L-Languages is Polynomially Complete. *Inf. Process. Lett.*, 3(5):138–143, 1975.

[vL76] Jan van Leeuwen. Variations of a New Machine Model. In *17th Annual Symposium on Foundations of Computer Science, Houston, Texas, USA, 25-27 October 1976*, pages 228–235. IEEE Computer Society, 1976.

[Vol90] Heribert Vollmer. The Gap-Language-Technique Revisited. In Egon Börger, Hans Kleine Büning, Michael M. Richter, and Wolfgang Schönfeld, editors, *Computer Science Logic, 4th Workshop, CSL '90, Heidelberg, Germany, October 1-5, 1990, Proceedings*, volume 533 of *Lecture Notes in Computer Science*, pages 389–399. Springer, 1990.

[Vol99] Heribert Vollmer. *Introduction to circuit complexity - a uniform approach.* Texts in theoretical computer science. Springer, 1999.