Reachability in Higher-Order-Counters ⋆

Alexander Heußner[1] and Alexander Kartzow[2]

[1] Otto-Friedrich-Universität Bamberg, Germany
[2] Universität Leipzig, Germany

Abstract. Higher-order counter automata (HOCA) can be either seen as a restriction of higher-order pushdown automata (HOPA) to a unary stack alphabet, or as an extension of counter automata to higher levels. We distinguish two principal kinds of HOCA: those that can test whether the topmost counter value is zero and those which cannot.
We show that control-state reachability for level k HOCA with 0-test is complete for $(k-2)$-fold exponential space; leaving out the 0-test leads to completeness for $(k-2)$-fold exponential time. Restricting HOCA (without 0-test) to level 2, we prove that global (forward or backward) reachability analysis is **P**-complete. This enhances the known result for pushdown systems which are subsumed by level 2 HOCA without 0-test.
We transfer our results to the formal language setting. Assuming that $\mathbf{P} \subsetneq \mathbf{PSPACE} \subsetneq \mathbf{EXPTIME}$, we apply proof ideas of Engelfriet and conclude that the hierarchies of languages of HOPA and of HOCA form strictly interleaving hierarchies. Interestingly, Engelfriet's constructions also allow to conclude immediately that the hierarchy of collapsible pushdown languages is strict level-by-level due to the existing complexity results for reachability on collapsible pushdown graphs. This answers an open question independently asked by Parys and by Kobayashi.

1 From Higher-Order Pushdowns to Counters and Back

Higher-order pushdown automata (HOPA) — also known as iterated pushdown automata — were first introduced by Maslov in [14] and [15] as an extension of classical pushdown automata where the pushdown storage is replaced by a nested pushdown of pushdowns of ... of pushdowns. After being originally studied as acceptors of languages, these automata have nowadays obtained renewed interest as computational model due to their connection to safe higher-order recursion schemes. Recent results focus on algorithmic questions concerning the underlying configuration graphs, e.g., Carayol and Wöhrle [5] showed decidability of the monadic second-order theories of higher-order pushdown graphs due to the pushdown graph's connection to the Caucal-hierarchy [6], and Hague and Ong determined the precise complexity of the global backwards reachability problem for HOPA: for level k it is complete for $\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k-1}(n^d))$ [9] .[3]

⋆ The second author is supported by the DFG research project GELO. We both thank M. Bojańczyk, Ch. Broadbent, and M. Lohrey for helpful discussions and comments.

[3] We define $\exp_0(n) := n$ and $\exp_{k+1}(n) := \exp(\exp_k(n))$ for any natural number k.

In the setting of classical pushdown automata it is well known that restricting the stack alphabet to one single symbol, i.e., reducing the pushdown storage to a counter, often makes solving algorithmic problems easier. For instance, control state reachability for pushdown automata is **P**-complete whereas it is $\mathbf{NSPACE}(\log(n))$-complete for counter automata. Then again, results from counter automata raise new insights to the pushdown case by providing algorithmic lower bounds and important subclasses of accepted languages separating different classes of complexity. In this paper we lift this idea to the higher-order setting by investigating reachability problems for higher-order counter automata (HOCA), i.e., HOPA over a one-element stack alphabet. Analogously to counter automata, we introduce level k HOCA in two variants: with or without 0-tests. Throughout this paper, we write $\mathsf{k\text{-}HOCA}^-$ for the variant without 0-tests and $\mathsf{k\text{-}HOCA}^+$ for the variant with 0-tests. Transferring our results' constructions back to HOPA will then allow to answer a recent open question [16,13].

To our knowledge, the only existing publication on HOCA is by Slaats [17]. She proved that $(\mathsf{k}+1)\text{-}\mathsf{HOCA}^+$ can simulate level k pushdown automata (abbreviated k-HOPA). In fact, even $(\mathsf{k}+1)\text{-}\mathsf{HOCA}^-$ simulate k-HOPA. Slaats conjectured that $L(\mathsf{k\text{-}HOCA}^+) \subsetneq L(\mathsf{k\text{-}HOPA})$ where $L(X)$ denotes the languages accepted by automata of type X. We can confirm this conjecture by combining the proof ideas of Engelfriet [7] with our main result on control-state reachability for HOCA in Theorems 13 and 14: control state reachability on $\mathsf{k\text{-}HOCA}^+$ is complete for $\mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$ and control state reachability on $\mathsf{k\text{-}HOCA}^-$ is complete for $\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$. These results are obtained by adapting a proof strategy relying on reductions to bounded space storage automata originally stated for HOPA by Engelfriet [7]. His main tool are auxiliary $\mathbf{SPACE}(b(n))$ P^k automata where P^k denotes the storage type of a k-fold nested pushdown (see Section 2 for a precise definition). Such a (two-way) automaton has an additional storage of type P^k, and a Turing machine worktape with space $b(n)$. His main technical result shows a trade off between the space bound b and the number of iterated pushdowns k. Roughly speaking, exponentially more space allows to reduce the number of nestings of pushdowns by one. Similarly, at the cost of another level of pushdown, one can trade alternation against nondeterminism. Here, we also restate reachability on $\mathsf{k\text{-}HOCA}^+$ as a membership problem on alternating auxiliary $\mathsf{SPACE}(\exp_{k-3}(n))$ $\mathcal{Z}+$ automata (where $\mathcal{Z}+$ is the new storage type of a counter with 0-test). For our $\mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$-hardness proof we provide a reduction of $\mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp(\exp_{k-3}(n^d)))$ to alternating auxiliary $\mathbf{SPACE}(\exp_{k-3}(n))$ $\mathcal{Z}+$ automata that is inspired by Jancar and Sawa's **PSPACE**-completeness proof for the non-emptiness of alternating automata [10]. For containment we adapt the proof of Engelfriet [7] and show that membership for alternating auxiliary $\mathbf{SPACE}(\exp_{k-3}(n))$ $\mathcal{Z}+$ automata can be reduced to alternating reachability on counter automata of size $\exp_{k-2}(n)$, where n is the size of the original input, which is known to be in $\mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$ (cf. [8]).

For the case of $\mathsf{k\text{-}HOCA}^-$ the hardness follows directly from the hardness of reachability for level $(k-1)$ pushdown automata and the fact that the latter can be simulated by $\mathsf{k\text{-}HOCA}^-$. For containment in $\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$ the mentioned machinery of Engelfriet reduces the problem to the case $k = 2$.

The proof that control-state reachability on $\mathsf{2\text{-}HOCA}^-$ is in $\mathbf{P}$ is implied by Theorem 5 which proves a stronger result: both the global regular forward and backward reachability problems for $\mathsf{2\text{-}HOCA}^-$ are $\mathbf{P}$-complete. The backward reachability problem asks, given a regular set C of configurations, for a (regular) description of all configurations that allow to reach one in C. This set is typically denoted as $\mathsf{pre}^*(C)$. Note that there is no canonical way of defining a regular set of configurations of $\mathsf{2\text{-}HOCA}^-$. We are aware of at least three possible notions: regularity via 2-store automata [2], via sequences of pushdown-operations [4], and via encoding in regular sets of trees. We stick to the latter, and use the encoding of configurations as binary trees introduced in [11]: We call a set C of configurations regular if the set of encodings of configurations $\{\mathsf{E}(c) \mid c \in C\}$ is a regular set of trees (where E denotes the encoding function from [11]). Note that the other two notions of regularity are both strictly weaker (with respect to expressive power) than the notion of regularity we use here. Nevertheless, our result does not carry over to these other notions of regularity as they admit more succinct representations of certain sets of configurations. See Appendix E for details.

Besides computing $\mathsf{pre}^*(C)$ in polynomial time our algorithm also allows to compute the reachable configurations $\mathsf{post}^*(C)$ in polynomial time. Thus, $\mathsf{2\text{-}HOCA}^-$ subsumes the well-known class of pushdown systems [1] while still possessing the same good complexity with respect to reachability problems.

2 Formal Model of Higher-Order Counters

2.1 Storage Types and Automata

An elegant way for defining HOCA and HOPA is the use of storage types and operators on these (following [7]). For simplicity, we restrict ourselves to what Engelfriet calls *finitely encoded* storage types.

Definition 1. *For X some set, we call a function $t : X \to \{true, false\}$ an X-*test *and a partial function $f : X \to X$ an X-*operation*.*
A storage type *is a tuple $\mathcal{S} = (X, T, F, x_0)$ where X is the set of $\mathcal{S}$-configurations, $x_0 \in X$ the initial $\mathcal{S}$-configuration, T a finite set of X-tests and F a finite set of X-operations containing the identity on X, i.e., $\mathsf{id}_X \in F$.*

Let us fix some finite alphabet Σ with a distinguished symbol $\perp \in \Sigma$. Let $\mathcal{P}_\Sigma = (X, T, F, x_0)$ be the *pushdown storage type* where $X = \Sigma^+$, $x_0 = \perp$, $T = \{\mathsf{top}_\sigma \mid \sigma \in \Sigma\}$ with $\mathsf{top}_\sigma(w) = true$ if $w \in \Sigma^*\sigma$, and $F = \{\mathsf{push}_\sigma \mid \sigma \in \Sigma\} \cup \{\mathsf{pop}, \mathsf{id}\}$ with $\mathsf{id} = \mathsf{id}_X$, $\mathsf{push}_\sigma(w) = w\sigma$ for all $w \in X$, and $\mathsf{pop}(w\sigma) = w$ for all $w \in \Sigma^+$ and $\sigma \in \Sigma$ and $\mathsf{pop}(\sigma)$ undefined for all $\sigma \in \Sigma$. Hence, $\mathcal{P}_\Sigma$ represents a classical pushdown stack over the alphabet Σ. We write $\mathcal{P}$ for $\mathcal{P}_{\{\perp,0,1\}}$.

We define the storage type *counter without 0-test* $\mathcal{Z} = \mathcal{P}_{\{\perp\}}$, which is the pushdown storage over a unary pushdown alphabet. We define the storage type *counter with* 0*-test* $\mathcal{Z}+$ exactly like $\mathcal{Z}$ but we add the test *empty*? to the set of tests where *empty*?$(x) = true$ if $x = \bot$ (the plus in $\mathcal{Z}+$ stands for "with 0-test"). In other words, *empty*? returns false iff the operation pop is applicable.

Definition 2. *For a storage type $\mathcal{S} = (X, T, F, x_0)$ we define an $\mathcal{S}$* automaton *as a tuple $\mathcal{A} = (Q, q_0, q_f, \Delta)$ where as usual Q is a finite set of states with initial state q_0 and final state q_f and Δ is the transition relation. The difference to a usual automaton is the definition of Δ by $\Delta = Q \times \{true, false\}^T \times Q \times F$.*

For $q \in Q$ and $x \in X$, a transition $\delta = (q, R, p, f)$ is applicable to the *configuration* (q, x) if $f(x)$ is defined and if for each test $t \in T$ we have $R(t) = t(x)$, i.e., the result of the storage-tests on the storage configuration x agree with the test results required by the transition δ. If δ is applicable, application of δ leads to the configuration $(p, f(x))$. The notions of a run, the accepted language, etc. are now all defined as expected.

The Pushdown Operator We also consider $\mathcal{P}_\Sigma$ as an operator on other storage types as follows. Given a storage type $\mathcal{S} = (X, T, F, x_0)$ let the storage type *pushdown of* $\mathcal{S}$ be $\mathcal{P}_\Sigma(\mathcal{S}) = (X', T', F', x'_0)$ where $X' = (\Sigma \times X)^+$, $x'_0 = (\bot, x_0)$, $T' = \{\mathsf{top}_\sigma \mid \sigma \in \Sigma\} \cup \{test(t) \mid t \in T\}$, $F' = \left\{\mathsf{push}_{\gamma,f} \mid \gamma \in \Sigma, f \in F\right\} \cup \left\{\mathsf{stay}_f \mid f \in F\right\} \cup \{pop\}$, and where for all $x' = \beta(\sigma, x)$, $\beta \in (\Sigma \times X)^*$, $\sigma \in \Sigma$, $x \in X$ it holds that

- $\mathsf{top}_\tau(x') = (\tau = \sigma)$,
- $test(t)(x') = t(x)$,
- $\mathsf{push}_{\tau,f}(x') = \beta(\sigma, x)(\tau, f(x))$
  if f is defined on x (and undefined otherwise),
- $\mathsf{stay}_f(x') = \beta(\sigma, f(x))$
  if f is defined on x (and undefined otherwise), and
- $\mathsf{pop}(x') = \beta$ if β is nonempty (and undefined otherwise).

Note that $\mathsf{stay}_{\mathsf{id}_X} = \mathsf{id}_{X'}$ whence F' contains the identity. As for storages, we define the operator $\mathcal{P}$ to be the operator $\mathcal{P}_{\{\bot,0,1\}}$.

2.2 HOPA, HOCA, and their Reachability Problems

We can define the iterative application of the operator $\mathcal{P}$ on some storage $\mathcal{S}$ as follows: let $\mathcal{P}^0(\mathcal{S}) = \mathcal{S}$ and $\mathcal{P}^{k+1}(\mathcal{S}) = \mathcal{P}(\mathcal{P}^k(\mathcal{S}))$. A *level k higher-order pushdown automaton* is a $\mathcal{P}^{k-1}(\mathcal{P})$ automaton. We abbreviate the class of all these automata with k-HOPA. A *level k higher-order counter automaton with zero-test* is a $\mathcal{P}^{k-1}(\mathcal{Z}+)$ automaton and k-HOCA$^+$ denotes the corresponding class.[4]. Similarly, k-HOCA$^-$ denotes the class of *level k higher-order counter automata without zero-test* which is the class of $\mathcal{P}^{k-1}(\mathcal{Z})$ automata. Obviously, for any level k it holds that $L(\text{k-HOCA}^-) \subseteq L(\text{k-HOCA}^+) \subseteq L(\text{k-HOPA})$ where $L(X)$ denotes the languages accepted by automata of type X.

We next define the reachability problems which we study in this paper.

[4] A priori our definition of k-HOCA$^+$ results in a stronger automaton model than that used by Slaats. In fact, both models are equivalent (cf. Appendix C).

Definition 3. *Given an $\mathcal{S}$ automaton and one of its control states $q \in Q$, then the* control state reachability problem *asks whether there is a configuration (q, x) that is reachable from (q_0, x_0) where $x \in X$ is an arbitrary $\mathcal{S}$-configuration.*

Assuming a notion of regularity for sets of $\mathcal{S}$ configurations (and hence for sets of configurations of $\mathcal{S}$ automata), we can also define a global variant of the control state reachability problem.

Definition 4. *Given an $\mathcal{S}$ automaton $\mathcal{A}$ and a regular set of configurations C, the* regular backwards reachability problem *demands a description of the set of configurations from which there is a path to some configuration $c \in C$.*

Analogously, the regular forward reachability problem asks for a description of the set of configurations reachable from a given regular set C. In the following section, we consider the regular backwards (and forwards) reachability problem for the class of 2-HOCA$^-$ only.

3 Regular Reachability for 2-HOCA$^-$

The goal of this section is to prove the following theorem extending a known result on regular reachability on pushdown systems to 2-HOCA$^-$:

Theorem 5. *Reg. backwards/forwards reachability on* 2-HOCA$^-$ *is* **P***-complete.*

3.1 Returns, Loops, and Control State Reachability

Proving Theorem 5 is based on the "returns-&-loops" construction for 2-HOPA of [11]. As a first step, we consider the simpler case of control-state reachability:

Proposition 6. *Control state reachability for* 2-HOCA$^-$ *is* **P***-complete.*

In [11] it has been shown that certain runs, so-called *loops* and *returns*, are the building blocks of any run of a 2-HOPA in the sense that solving a reachability problem amounts to deciding whether certain loops and returns exist. Here, we analyse these notions more precisely in the context of 2-HOCA$^-$ in order to derive a polynomial control state reachability algorithm. Using this algorithm we can then also solve the regular backwards reachability problem efficiently.

For this section, we fix a $\mathcal{P}(\mathcal{Z})$-automaton $\mathcal{A} = (Q, q_0, F, \Delta)$. Recall that the $\mathcal{P}(\mathcal{Z})$-configurations of $\mathcal{A}$ are elements of $(\Sigma \times \{\bot\}^+)^+$. We identify $\bot^{m+1}$ with the natural number m and the set of storage configurations with $(\Sigma \times \mathbb{N})^+$.

Definition 7. *Let $s \in (\Sigma \times \mathbb{N})^+$, $t \in \Sigma \times \mathbb{N}$ and $q, q' \in Q$ be states of $\mathcal{A}$. A* return *of $\mathcal{A}$ from (q, st) to (q', s) is a run r from (q, st) to (q', s) such that except for the final configuration no configuration of r is in $Q \times \{s\}$.*

Let $s \in (\Sigma \times \mathbb{N})^*$, $t \in \Sigma \times \mathbb{N}$. A loop *of $\mathcal{A}$ from (q, st) to (q', st) is a run r from (q, st) to (q', st) such that no configuration of r is in $Q \times \{s\}$.*

One of the underlying reasons why control state reachability for pushdown systems can be efficiently solved is the fact that it is always possible to reach a certain state without increasing the pushdown by more than polynomially many elements. In the following, we prove an analogue of this fact for $\mathcal{P}(\mathcal{Z})$. For a given configuration, if there is a return or loop starting in this configuration, then this return or loop can be realised without increasing the (level 2) pushdown more than polynomially. This is due to the monotonic behaviour of $\mathcal{Z}$: given a $\mathcal{Z}$ configuration x, if we can apply a sequence φ of transitions to x then we can apply φ to all bigger configurations, i.e., to any configuration of the form $\mathsf{push}_{\perp}^{n}(x)$. Note that this depends on the fact that $\mathcal{Z}$ contains only trivial tests (the test $\mathsf{top}_{\perp}$ always returns true). In contrast, for $\mathcal{Z}+$, if φ applies a couple of pop operations and then tests for zero and performs a transition, then this is not applicable to a bigger counter because the 0-test would now fail.

For a $\mathcal{P}(\mathcal{Z})$ configuration $x = (\sigma_1, n_1)(\sigma_2, n_2)\dots(\sigma_m, n_m)$, let $|x| = m$ be its *height*. Let r be some run starting in (q, x) for some $q \in Q$. The run r *increases the height by at most* k if $|x'| \leq |x| + k$ for all configurations (q', x') of r.

Definition 8. *Let $s \in (\{\perp\} \times \mathbb{N})^+$. We write $\mathsf{ret}_k(s)$ and $\mathsf{lp}_k(s)$, resp., for the set of pairs of initial and final control states of returns or loops starting in s and increasing the height by at most k. We write $\mathsf{ret}_\infty(s)$ and $\mathsf{lp}_\infty(s)$,resp., for the union of all $\mathsf{ret}_k(sw)$ or $\mathsf{lp}_k(s)$.*

The existence of a return (or loop) starting in sw (or $s'w$) (with $s \in (\{\perp\} \times \mathbb{N})^+, s' \in (\{\perp\} \times \mathbb{N})^*$ and $w \in \{\perp\} \times \mathbb{N}$) does not depend on the concrete choice of s or s'. Thus, we also write $\mathsf{ret}_k(w)$ for $\mathsf{ret}_k(sw)$ and $\mathsf{lp}_k(w)$ for $\mathsf{lp}_k(s'w)$.

By induction on the length of a run, we first prove that $\mathcal{P}(\mathcal{Z})$ is *monotone* in the following sense: let $s \in (\Sigma \times \mathbb{N})^*, t = (\sigma, n) \in \Sigma \times \mathbb{N}$, $q, q' \in Q$ and r a run starting in (q, st) and ending in state q'. If the topmost counter of each configuration of r is at least m, then for each $n' \geq n - m$ there is a run r' starting in $(q, s(\sigma, n'))$ and performing exactly the same transitions as r. In particular, for all $k \in \mathbb{N} \cup \{\infty\}$, $\sigma \in \Sigma$ and $m_1 \leq m_2 \in \mathbb{N}$, $\mathsf{ret}_k((\sigma, m_1)) \subseteq \mathsf{ret}_k((\sigma, m_2))$ and $\mathsf{lp}_k((\sigma, m_1)) \subseteq \mathsf{lp}_k((\sigma, m_2))$.

We next show that the sequence $(\mathsf{ret}_k((\sigma, m)))_{m \in \mathbb{N}}$ stabilises at $m = |\Sigma||Q|^2$. From this we conclude that $\mathsf{ret}_\infty = \mathsf{ret}_{|\Sigma|^2|Q|^4}$, i.e., in order to realise a return with arbitrary fixed initial and final configuration, we do not have to increase the height by more than $|\Sigma|^2|Q|^4$ (if there is such a return at all).

Lemma 9. *For $k \in \mathbb{N} \cup \{\infty\}$, $\sigma \in \Sigma$, $m \geq |\Sigma||Q|^2$, and $m' \geq 2 \cdot |\Sigma||Q|^2$, we have $\mathsf{ret}_k((\sigma, m)) = \mathsf{ret}_k((\sigma, |\Sigma||Q|^2))$ and $\mathsf{lp}_\infty((\sigma, m')) = \mathsf{lp}_\infty((\sigma, 2 \cdot |\Sigma||Q|^2))$.*

The proof uses the fact that we can find an $m' \leq |\Sigma||Q|^2$ with $\mathsf{ret}_k((\sigma, m')) = \mathsf{ret}_k(\sigma, m' + 1)$ for all σ by the pigeonhole-principle. Using monotonicity of $\mathcal{P}(\mathcal{Z})$ we conclude that $\mathsf{ret}_k(\sigma, m') = \mathsf{ret}_k(\sigma, m)$ for all $m \geq m'$. A similar application of the pigeonhole-principle shows that there is a $k \leq |\Sigma|^2 \cdot |Q|^4$ such that $\mathsf{ret}_k((\sigma, i)) = \mathsf{ret}_{k+1}((\sigma, i))$ for all σ and all $i \leq |\Sigma||Q|^2$ (or equivalently for all $i \in \mathbb{N}$). By induction on $k' \geq k$ we show that $\mathsf{ret}_{k'} = \mathsf{ret}_k$ because any subreturn that increases the height by $k + 1$ can be replaced by a subreturn that only increases the height by k. Thus, we obtain the following lemma.

```
GeneratePDA(𝒜, A):
Input: 2-HOCA⁻ 𝒜 = (Q, q0, Δ) over Σ, matrix A = (a_{σ,p,q})_{(σ,p,q)∈Σ×Q²} over ℕ ∪ {∞}
Output: 1-HOPA 𝒜' simulating 𝒜

k0 := |Σ|²·|Q|⁴; h0 :=|Σ|·|Q²|; Δ' := ∅
foreach δ ∈ Δ:
   if δ == (q, (σ, ⊥), stay_pop, p):
      foreach i in {0, ..., h0}: Δ' := Δ' ∪ {((q, σ), ⊥_i, pop, (p, σ)), ((q, σ), , ⊥_∞, pop, (p, σ))}
   elseif δ==(q, (σ, ⊥), stay_push⊥, p)
      Δ' := Δ' ∪ {((q, σ), ⊥_∞, push_⊥∞, (p, σ))} ∪ {((q, σ), ⊥_h0, push_⊥∞, (p, σ))}
      foreach i in {0, ..., h0 − 1}}: Δ' := Δ' ∪ {((q, σ), ⊥_i, push_⊥i+1, (p, σ))}
   elseif δ==(q, (σ, ⊥), push_τ,id, p)
      foreach r in Q such that a_{τ,p,r} ≠ ∞:
         foreach i in {a_{τ,p,r}, a_{τ,p,r} + 1, ..., h0} ∪ {∞}: Δ' := Δ' ∪ {((q, σ), ⊥_i, id, (r, σ))
𝒜' := (Q × Σ, (q0, ⊥), Δ')
return 𝒜'
```

Fig. 1. 2-HOCA$^-$ to 1-HOPA Reduction Algorithm

Lemma 10. *For all $i \in \mathbb{N}$ and $\sigma \in \Sigma$, we have $\mathsf{ret}_\infty((\sigma, i)) = \mathsf{ret}_{|\Sigma|^2 \cdot |Q|^4}((\sigma, i))$ and $\mathsf{lp}_\infty = \mathsf{lp}_{|\Sigma|^2 |Q|^4 + 1}$.*

We now can prove that control-state reachability on 2-HOCA$^-$ is **P**-complete.

Proof (of Proposition 6). Since 2-HOCA$^-$ can trivially simulate pushdown automata, hardness follows from the analogous hardness result for pushdown automata. Containment in **P** uses the following ideas:

1. We assume that the input $(\mathcal{A}, q)$ satisfies that q is reachable in $\mathcal{A}$ iff $(q, (\perp, 0))$ is reachable and that $\mathcal{A}$ only uses instructions of the forms pop, $\mathsf{push}_{\sigma,\mathsf{id}}$, and stay_f. Given any 2-HOCA$^-$ $\mathcal{A}'$ and a state q, it is straightforward to construct (in polynomial time) a 2-HOCA$^-$ $\mathcal{A}$ that satisfies this condition such that q is reachable in $\mathcal{A}'$ iff it is reachable in $\mathcal{A}$.
2. Recall that $\mathsf{ret}_\infty(w) = \mathsf{ret}_{k_0}(w)$ for $k_0 = |\Sigma|^2 \cdot |Q|^4$ and for all $w \in \Sigma \times \mathbb{N}$. Set $h_0 = |\Sigma| \cdot |Q^2|$. We want to compute a table $(a_{\sigma,p,q})_{\sigma,p,q \in \Sigma \times Q^2}$ with values in $\{\infty, 0, 1, 2, \ldots, h_0\}$ such that $a_{\sigma,p,q} = \min\{i \mid (p, q) \in \mathsf{ret}_{k_0}((\sigma, i))\}$ (where we set $\min\{\emptyset\} = \infty$). Due to Lemmas 9 and 10 such a table represents ret_∞ in the sense that $(p, q) \in \mathsf{ret}_\infty((\sigma, i))$ iff $i \geq a_{\sigma,p,q}$.
3. With the help of the table $(a_{\sigma,p,q})_{(\sigma,p,q) \in \Sigma \times Q^2}$ we compute in polynomial time a $\mathcal{P}$ automaton $\mathcal{A}_\infty$ which executes the same level 1 transitions as $\mathcal{A}$ and simulates loops of $\mathcal{A}$ in the following sense: if there is a loop of $\mathcal{A}$ starting in $(q, (\sigma, i))$ performing first a $\mathsf{push}_{\tau,\mathsf{id}}$ operation and then performing a return with final state p, we allow $\mathcal{A}'$ to perform an id-transition from $(q, (\sigma, i))$ to $(p, (\sigma, i))$. This new system basically keeps track of the height of the pushdown up to h_0 by using a pushdown alphabet $\{\perp_0, \ldots, \perp_{h_0}, \perp_\infty\}$ where the topmost symbol of the pushdown is $\perp_i$ iff the height of the pushdown is i (where ∞ stands for values above h_0). After this change of pushdown alphabet, the additional id-transitions are easily computable from the table $(a_{\sigma,p,q})_{(\sigma,p,q) \in \Sigma \times Q^2}$. The resulting system has size $O(h_0^2 \cdot (|\mathcal{S}| + 1))$, i.e., is polynomial in the original system $\mathcal{A}$.

ReachHOCA-$(\mathcal{A}, q_f)$:
Input: 2-HOCA$^-$ $\mathcal{A} = (Q, q_0, \Delta)$ over Σ, $q_f \in Q$
Output: whether q_f is reachable in $\mathcal{A}$

k_0 := $|\Sigma|^2 \cdot |Q|^4$; h_0 :=$|\Sigma| \cdot |Q^2|$;
foreach (σ, p, q) in $\Sigma \times Q^2$: $a_{\sigma,p,q} := \infty$
for $k = 1, 2, \dots, k_0$:
    $\mathcal{A}_k :=$ GeneratePDA$(\mathcal{A}, (a_{\sigma,p,q})_{(\sigma,p,q)\in\Sigma\times Q^2})$
    foreach $(r, (\tau, \bot), \mathsf{pop}, q)$ in Δ and (σ, p) in $\Sigma \times Q$:
        for $i = h_0, h_0 - 1, \dots, 1, 0$:
        if ReachPDA$(\mathcal{A}_k, ((p, \sigma), i), (r, \tau))$: $a'_{\sigma,p,q} := i$
    foreach (σ, p, q) in $\Sigma \times Q^2$: $a_{\sigma,p,q}$:=$a'_{\sigma,p,q}$
$\mathcal{A}_\infty$:=GeneratePDA$(\mathcal{A}, (a_{\sigma,p,q})_{(\sigma,p,q)\in\Sigma\times Q^2})$
if Reach$(\mathcal{A}_\infty, ((q_0, \bot), 0), (q_f, \bot))$: **return** true **else return** false

Fig. 2. Reachability on 2-HOCA$^-$ Algorithm 2

4. Using [1], check for reachability of q in the pushdown automaton $\mathcal{A}_\infty$.

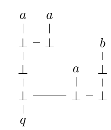
In fact, for step 2 we already use a variant of steps 3 and 4: we compute $\mathsf{ret}_\infty = \mathsf{ret}_{|\Sigma|^2|Q|^4}$ by induction starting with ret_0. If we remove all level 2 operations from $\mathcal{A}$ and store the topmost level 2 stack-symbol in the control state we obtain a pushdown automaton $\mathcal{B}$ such that $(q, q') \in \mathsf{ret}_0(\sigma, k)$ (w.r.t. $\mathcal{A}$) iff there is a transition $(p, (\sigma, \bot), \mathsf{pop}, q')$ of $\mathcal{A}$ and the control state (p, σ) is reachable from $((p, \sigma), k)$ in $\mathcal{B}$. Thus, the results of polynomially many reachability queries for $\mathcal{B}$ determine the table for ret_0. Similarly, we can use the table of ret_i to compute the table of ret_{i+1} as follows. A return extending the height of the pushdown by $i+1$ decomposes into parts that do not increase the height at all and parts that perform a $\mathsf{push}_{\tau,\mathsf{id}}$ followed by a return increasing the height by at most i. Using the table for ret_i we can easily enrich $\mathcal{B}$ by id-transitions that simulate such push operations followed by returns increasing the height by at most i. Again, determining whether $(q, q') \in \mathsf{ret}_{i+1}(\sigma, k)$ reduces to one reachability query on this enriched $\mathcal{B}$ for each pop-transition of $\mathcal{A}$.

With these ideas in mind, it is straightforward to check that algorithm ReachHOCA- in Figure 2 (using algorithm GeneratePDA of Figure 1 as subroutine for step 3) solves the reachability problem for 2-HOCA$^-$ (of the form described in step 1) in polynomial time. In this algorithm, ReachPDA $(\mathcal{A}', c, q)$ refers to the classical polynomial time algorithm that determines whether in the (level 1) pushdown automaton $\mathcal{A}'$ state q is reachable when starting in configuration c; a transition $(q, (\sigma, \tau), f, p)$ refers to a transition from state q to state p applying operation f that is executable if the (level 2) test top_σ and the (level 1) test $test(\mathsf{top}_\tau)$ both succeed. $\square$

3.2 Regular Reachability

In order to define regular sets of configurations, we recall the encoding of 2-HOPA configurations as trees from [11]. Let $p = (\sigma_1, v_1)(\sigma_2, v_2)\dots(\sigma_m, v_m) \in \mathcal{P}(\mathcal{Z})$. If $v_1 = 0$, we set $p_l = \emptyset$ and $p_r = (\sigma_2, v_2)\dots(\sigma_m, v_m)$. Otherwise, there is a maximal $1 \leq j \leq m$ such that $v_1, \ldots, v_j \geq 1$ and we set $p_l = (\sigma_1, v_1 - 1) \ldots (\sigma_j, v_j - 1)$ and $p_r = (\sigma_{j+1}, v_{j+1}) \ldots (\sigma_m, v_m)$ if $j < m$ and $p_r = \emptyset$ if $j = m$. The *tree-encoding* E of p is given as follows:

$$\mathsf{E}(p) = \begin{cases} \emptyset & \text{if } p = \emptyset \\ \bot(\sigma_1, \mathsf{E}(p_r)) & \text{if } p = (\sigma_1, 0)p_r \\ \bot(\mathsf{E}(p_l), \mathsf{E}(p_r)) & \text{otherwise,} \end{cases}$$



where $\bot(t_1, t_2)$ is the tree with root $\bot$ whose left subtree is t_1 and whose right subtree is t_2. For a configuration $c = (q, p)$ we define $\mathsf{E}(c)$ to be the tree $q(\mathsf{E}(p), \emptyset)$. The picture beside the definition of E shows the encoding of the configuration $(q, (a, 2)(a, 2)(a, 0)(b, 1))$. Note that for each element (σ, i) of p, there is a path to a leaf l which is labelled by σ such that the path to l contains $i + 2$ left successors. Moreover, the inorder traversal of the tree induces an order of the leaves which corresponds to the left-to-right order of the elements of p. We call a set C of configurations *regular* if the set $\{\mathsf{E}(c) \mid c \in C\}$ is a regular set of trees.

E turns the reachability predicate on 2-HOCA^- into a tree-automatic relation [11], i.e., for a given 2-HOCA^- $\mathcal{A}$, there is a tree-automaton $\mathcal{T}_\mathcal{A}$ accepting the convolution of $\mathsf{E}(c_1)$ and $\mathsf{E}(c_2)$ for 2-HOCA^- configurations c_1 and c_2 iff there is a run of $\mathcal{A}$ from c_1 to c_2. This allows to solve the regular backwards reachability problem as follows. On input a 2-HOCA^- and a tree automaton $\mathcal{T}$ recognising a regular set C of configurations, we first compute the tree-automaton $\mathcal{T}_\mathcal{A}$. Then using a simple product construction of $\mathcal{T}_\mathcal{A}$ and $\mathcal{T}$ and projection, we obtain an automaton $\mathcal{T}_{\mathsf{pre}}$ which accepts $\mathsf{pre}^*(C) = \{\mathsf{E}(c) \mid \exists c' \in C \text{ and a run from } c \text{ to } c'\}$. The key issue for the complexity of this construction is the computation of $\mathcal{T}_\mathcal{A}$ from $\mathcal{A}$. The explicit construction of $\mathcal{T}_\mathcal{A}$ in [11] involves an exponential blow-up. In this construction the blow-up is only caused by a part of $\mathcal{T}_\mathcal{A}$ that computes $\mathsf{ret}_\infty(\sigma, m)$ for each $\sigma \in \Sigma$ on input a path whose labels form the word $\bot^m$. Thus, we can exhibit the following consequence.

Corollary 11 ([11]). *Given a* 2-HOCA^- $\mathcal{A}$ *with state set* Q*, we can compute the tree automaton* $\mathcal{T}_\mathcal{A}$ *in* $\mathbf{P}$*, if we can compute from* $\mathcal{A}$ *in* $\mathbf{P}$ *a deterministic word automaton* $\mathcal{T}'$ *with state set* $Q' \subseteq \prod_{\sigma \in \Sigma} (2^{Q \times Q})^2$ *such that for all* $m \in \mathbb{N}$ *the state of* $\mathcal{T}'$ *on input* $\bot^m$ *is* $(\mathsf{ret}_\infty(\sigma, m), \mathsf{lp}_\infty(\sigma, m))_{\sigma \in \Sigma}$.

Thus, the following lemma completes the proof of Theorem 5.

Lemma 12. *Let* $\mathcal{A}$ *be a* 2-HOCA^- *with state set* Q*. We can compute in polynomial time a deterministic finite word automaton* $\mathcal{A}'$ *with state set* Q' *of size at most* $2 \cdot (|\Sigma| \cdot |Q|^2 + 1)$ *such that* $\mathcal{A}'$ *is in state* $(\mathsf{ret}_\infty((\sigma, n)), \mathsf{lp}_\infty((\sigma, n)))_{\sigma \in \Sigma}$ *after reading* $\bot^n$ *for every* $n \in \mathbb{N}$.

Proof. Let $n_0 = 2 \cdot |\Sigma| \cdot |Q|^2$. Recall algorithm `ReachHOCA-` of Figure 2. In this polynomial time algorithm we computed a matrix $A = (a_{\sigma,p,q})_{(\sigma,p,q) \in \Sigma \times Q^2}$ representing ret_∞ and a pushdown automaton $\mathcal{A}_\infty$ (of level 1) simulating $\mathcal{A}$ in the sense that $\mathcal{A}_\infty$ reaches a configuration $((q, \sigma)p)$ for a pushdown p of height n if and only if $\mathcal{A}$ reaches $(q, (\sigma, n))$. It is sufficient to describe a polynomial time algorithm that computes $M_i = (\mathsf{ret}_\infty((\sigma, n)), \mathsf{lp}_\infty((\sigma, n)))_{\sigma \in \Sigma}$ for all $n \leq n_0$. $\mathcal{A}'$

is then the automaton with state set $\{M_i \mid i \leq n_0\}$, transitions from M_i to M_{i+1} for each $i < n_0$ and a transition from M_{n_0} to M_{n_0}. The correctness of this construction follows from Lemma 9.

Let us now describe how to compute M_i in polynomial time. Since $\mathcal{A}_\infty$ simulates $\mathcal{A}$ correctly, there is a loop from $(q, (\sigma, i))$ to $(q', (\sigma, i))$ of $\mathcal{A}$ if and only if there is a run of $\mathcal{A}_\infty$ from $((q,\sigma), p_i)$ to $((q',\sigma), p_i)$ for $p_i = \bot_0\bot_1 \dots \bot_i$ (where we identify $\bot_j$ with $\bot_\infty$ for all $j > h_0$). Thus, we can compute the loop part of M_i by n_0 many calls to an algorithm for reachability on pushdown systems. Note that $(p, q) \in \mathsf{ret}_\infty((\sigma, i))$ with respect to $\mathcal{A}$ if there is a state r and some $\tau \in \Sigma$ such that $(r, \tau, \mathsf{pop}, q)$ is a transition of $\mathcal{A}$ and (r, τ) is reachable in $\mathcal{A}_\infty$ from $((q, \sigma), i)$. Thus, with a loop over all transitions of $\mathcal{A}$ we reduce the computation of the returns component of M_i to polynomially many control state reachability problems on a pushdown system. $\square$

4 Reachability for $\mathsf{k\text{-}HOCA}^-$ and $\mathsf{k\text{-}HOCA}^+$

Using slight adaptations of Engelfriet's seminal paper [7], we can lift the result on reachability for $\mathsf{2\text{-}HOCA}^-$ to reachability for $\mathsf{k\text{-}HOCA}^-$ (cf. Appendix B).

Theorem 13. *For $k \geq 2$, the control state reachability problem for $\mathsf{k\text{-}HOCA}^-$ is complete for $\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$. For $k \geq 1$, the alternating control state reachability problem for $\mathsf{k\text{-}HOCA}^-$ is complete for $\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k-1}(n^d))$.*

Hardness follows from the hardness of control state reachability for $\mathsf{(k-1)\text{-}HOPA}$ [7] and the trivial fact that the storage type $\mathcal{P}^{k-1}$ of $\mathsf{(k-1)\text{-}HOPA}$ can be trivially simulated by the storage type $\mathcal{P}^{k-1}(\mathcal{Z})$ of $\mathsf{k\text{-}HOCA}^-$. Containment for the first claim is proved by induction on k (the base case $k = 2$ has been proved in the previous section). For $k \geq 3$, we use Lemma 7.11, Theorems 2.2 and 2.4 from [7] and reduce reachability of $\mathsf{k\text{-}HOCA}^-$ to reachability on (exponentially bigger) $\mathsf{(k-1)\text{-}HOCA}^-$. For the second claim, we adapt Engelfriet's Lemma 7.11 to a version for the setting of alternating automata (instead of nondeterministic automata) and use his Theorems 2.2. and 2.4 in order to show equivalence (up to logspace reductions) of alternating reachability for $\mathsf{(k-1)\text{-}HOCA}^-$ and reachability for $\mathsf{k\text{-}HOCA}^-$.

We can also reduce reachability for $\mathsf{k\text{-}HOCA}^+$ to reachability for $(k-1)$-fold exponentially bigger $\mathsf{1\text{-}HOCA}^+$. Completeness for $\mathbf{NSPACE}(\log(n))$ of reachability for $\mathsf{1\text{-}HOCA}^+$ (cf. [8]) yields the upper bounds for reachability for $\mathsf{k\text{-}HOCA}^+$. The corresponding lower bounds follow by applications of Engelfriet's theorems and an adaptation of the **PSPACE**-hardness proof for emptiness of alternating finite automata by Jancar and Sawa [10].

Theorem 14. *For $k \geq 2$, (alternating) control state reachability for $\mathsf{k\text{-}HOCA}^+$ is complete for ($\mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_{k-1}(n^d))$) $\mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$.*

5 Back to HOPS: Applications to Languages

Engelfriet [7] also discovered a close connection between the complexity of the control state reachability problem for a class of automata and the class of languages recognised by this class. We restate a slight extension (cf. AppendixD) of these results and use them to confirm Slaat's conjecture from [17].

Proposition 15. *Let $\mathcal{S}_1$ and $\mathcal{S}_2$ be storage types and C_1, C_2 complexity classes such that $C_1 \subsetneq C_2$. If control state reachability for nondeterministic $\mathcal{S}_i$ automata is complete for C_i, then there is a deterministic $\mathcal{S}_2$ automaton accepting some language L such that no nondeterministic $\mathcal{S}_1$-automaton accepts L.*

In fact, Engelfriet's proof can be used to derive a separating language. For a storage type $\mathcal{S} = (X, T, F, x_0)$, we define the *language of valid storage sequences* $\mathsf{VAL}(\mathcal{S})$ as follows. For each test $t \in T$ and $r \in \{true, false\}$ we set $t_r := \mathsf{id}\restriction_{\{x \in X \mid t(x)=r\}}$ and set $\Sigma = F \cup \{t_r \mid t \in T, r \in \{true, false\}\}$. For $s \in \Sigma^*$ such that $s = a_1 \dots a_n$, and $x \in X$ we write $s(x)$ for $a_n(a_{n-1}(\dots a_1(x)\dots))$. We define $\mathsf{VAL}(\mathcal{S}) = \{s \in \Sigma^* \mid s(x_0) \text{ is defined}\}$.

If the previous proposition separates the languages of $\mathcal{S}_2$ automata from those of $\mathcal{S}_1$ automata, then it follows from the proof that $\mathsf{VAL}(\mathcal{S}_2)$ is not accepted by any $\mathcal{S}_1$ automato (cf. AppendixD).

Corollary 16. *If* $\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_k(n^d)) \subsetneq \mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_k(n^d)) \subsetneq \mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k+1}(n^d))$, *then* $L((\mathsf{k}-1)\text{-}\mathsf{HOPA}) \subsetneq L(\mathsf{k}\text{-}\mathsf{HOCA}^-) \subsetneq L(\mathsf{k}\text{-}\mathsf{HOCA}^+) \subsetneq L(\mathsf{k}\text{-}\mathsf{HOPA})$.

The crucial underlying construction detail of the proof of Proposition 15 is quite hidden within the details of Engelfriet's technical and long paper. Its usefulness in other contexts — e.g., for higher-order pushdowns or counters — has been overseen so far. Here we give another application to collapsible pushdown automata: reachability for collapsible pushdown automata of level k is $\mathbf{DSPACE}(\exp_{k-1}(n))$-complete (cf. [3]). Thus, Proposition 15 trivially shows that the language of valid level $(k+1)$ collapsible pushdown storage sequences separates the collapsible pushdown languages of level $k+1$ from those of level k. This answers a question asked by several experts in this field (cf. [16,13]). In fact, [16] uses a long and technical construction to prove the weaker result that there are more level $2k$ collapsible pushdown languages than level k collapsible pushdown languages. From Proposition 15 one also easily derives the level-by-level strictness of the collapsible pushdown tree hierarchy and the collapsible pushdown graph hierarchy (cf. [12,13]).

6 Future Work

Our result on regular reachability gives hope that also complexity results on model checking for logics like the μ-calculus extend from pushdown automata to 2-HOCA. 2-HOCA$^-$ probably is a generalisation of pushdown automata that retains the good complexity results for basic algorithmic questions. It is also interesting whether the result on regular reachability extends to the different notions of regularity for k-HOCA mentioned in the introduction. HOCA also can be seen as a new formalism in the context of register machines as currently used in the verification of concurrent systems. HOCA allow to store pushdown-like structures of register values and positive results on model checking HOCA could be transferred to verification questions in this concurrent setting.

References

1. A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. of CONCUR'97*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.
2. A. Bouajjani and A. Meyer. Symbolic reachability analysis of higher-order context-free processes. In *Proc. of FSTTCS'04*, volume 3328 of *LNCS*, pages 135–147. Springer, 2004.
3. C. H. Broadbent, A. Carayol, M. Hague, and O. Serre. A saturation method for collapsible pushdown systems. In *ICALP (2)*, volume 7392 of *LNCS*, pages 165–176. Springer, 2012.
4. A. Carayol. Regular sets of higher-order pushdown stacks. In *Proc. of MFCS'05*, pages 168–179, 2005.
5. A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *Proc. of FSTTCS 2003*, volume 2914 of *LNCS*, pages 112–123. Springer, 2003.
6. D. Caucal. On infinite terms having a decidable monadic theory. In *MFCS 02*, pages 165–176, 2002.
7. J. Engelfriet. Iterated stack automata and complexity classes. *Inf. Comput.*, 95(1):21–75, 1991.
8. S. Göller. Reachability on prefix-recognizable graphs. *Inf. Process. Lett.*, 108(2):71–74, 2008.
9. M. Hague and C.-H. L. Ong. Symbolic backwards-reachability analysis for higher-order pushdown systems. *LMCS*, 4(4), 2008.
10. P. Jancar and Z. Sawa. A note on emptiness for alternating finite automata with a one-letter alphabet. *Inf. Process. Lett.*, 104(5):164–167, 2007.
11. A. Kartzow. Collapsible pushdown graphs of level 2 are tree-automatic. *Logical Methods in Computer Science*, 9(1), 2013.
12. A. Kartzow and P. Parys. Strictness of the collapsible pushdown hierarchy. In *MFCS*, volume 7464 of *LNCS*, pages 566–577, 2012.
13. N. Kobayashi. Pumping by typing. In *to appear in Proc. LICS*, 2013.
14. A. N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Sov. Math., Dokl.*, 15:1170–1174, 1974.
15. A. N. Maslov. Multilevel stack automata. *Problems of Information Transmission*, 12:38–43, 1976.
16. P. Parys. Variants of collapsible pushdown systems. In *Proc. of CSL'12*, volume 16 of *LIPIcs*, pages 500–515, 2012.
17. M. Slaats. *Infinite regular games in the higher-order pushdown and the parametrized setting*. PhD thesis, RWTH Aachen, 2012.

NOTE THAT THERE IS AN ADDITIONAL BIBLIOGRAPHIC REFERENCE SECTION AT THE END OF THE APPENDIX; WE USE CAPITAL LETTERS TO REFER TO THESE WORKS, E.G., [A].

A Omitted Proofs

If r is a run with domain $\{0, 1, \ldots, m\}$ and $0 \leq i \leq j \leq m$ we write $r\restriction_{[i,j]}$ for the subrun from position i to j, i.e., for the run r' with domain $\{0, 1, \ldots, j-i\}$ such that $r'(k) = r(i+k)$ for all $0 \leq k \leq j-i$.

Proof (of Lemma 9). We first prove the claim for returns. Set $\hat{m} = |\Sigma||Q|^2$. By induction on k, there is a $m_k \leq \hat{m}$ such that for all $k' \leq k$, all $\sigma \in \Sigma$ and all $m \geq m_k$ $\mathsf{ret}_{k'}((\sigma, m)) = \mathsf{ret}_{k'}((\sigma, m_k))$ and $\left|\bigsqcup_{\sigma\in\Sigma} \mathsf{ret}_k((\sigma, m_k))\right| \geq m_k$.[5]

For the base case $k = -1$, let $m_{-1} = 0$ and $\mathsf{ret}_{-1}((\sigma, m) = \emptyset$ for all $\sigma \in \Sigma$ and $m \in \mathbb{N}$.

For the induction step note that for every $i \in \mathbb{N}$

1. $\bigsqcup_{\sigma\in\Sigma} \mathsf{ret}_{k+1}((\sigma, i))$ contains at most $|\Sigma| \cdot |Q|^2$ many elements, and
2. $\bigsqcup_{\sigma\in\Sigma} \mathsf{ret}_k(\sigma, m_k) \subseteq \bigsqcup_{\sigma\in\Sigma} \mathsf{ret}_{k+1}(\sigma, m_k)$.

Since these sets are monotone in i, there is a minimal number $m_k \leq m_{k+1} \leq \hat{m}$ such that $\mathsf{ret}_{k+1}((\sigma, m_{k+1})) = \mathsf{ret}_{k+1}((\sigma, m_{k+1}+1))$ and $\bigsqcup_{\sigma\in\Sigma} \mathsf{ret}_{k+1}((\sigma, m_{k+1}))$ contains at least m_{k+1} elements.

In order to complete our proof, we have to show that for all $n > m_{k+1} + 1$, $\mathsf{ret}_{k+1}((\sigma, n)) \setminus \mathsf{ret}_{k+1}((\sigma, m_{k+1})) = \emptyset$. Heading for a contradiction, assume that there is a minimal n and a return r witnessing that $(q, q') \in \mathsf{ret}_{k+1}((\sigma, n)) \setminus \mathsf{ret}_{k+1}((\sigma, m_{k+1}))$.

1. If r never visits a configuration of the form $r(j) = (q_j, s_j(\sigma_j, 0))$, we obtain by monotonicity of $\mathcal{P}(\mathcal{Z})$ a run r' witnessing $(q, q') \in \mathsf{ret}_{k+1}((\sigma, n-1)) \setminus \mathsf{ret}_{k+1}((\sigma, m_{k+1}))$ contradicting minimality of n.
2. Otherwise, there is a maximal j such that $r(j) = (q_j, s_j(\sigma_j, 0))$. Since any operation alters the value of the topmost counter by at most 1, we find a maximal $j_1 \leq j$ such that $r(j_1) = (q_{j_1}, s_{j_1}(\sigma_{j_1}, m_{k+1}+1))$. Since r is a return and j_1 is not the last position in r, there is a $j_2 \geq j$ such that $r(j_2) = (q_{j_2}, s_{j_1})$, i.e., the restriction of r to $[j_1, j_2]$ is a return witnessing $(q_{j_1}, q_{j_2}) \in \mathsf{ret}_{k'}((\sigma_{j_1}, m_{k+1}+1)) = \mathsf{ret}_{k'}((\sigma_{j_1}, m_{k+1}))$ for some $k' \leq k+1$. Due to monotonicity of $\mathcal{P}(\mathcal{Z})$, we can lift a return from $(q_{j_1}, s_{j_1}(\sigma_{j_1}, m_{k+1}))$ to state q_{j_2} to a return r' from $(q_{j_1}, s_{j_1}(\sigma_{j_1}, m_{k+1}+1))$ to state q_{j_2} such that the topmost counter of all configurations are at least 1. Now replace in r the subrun $r\restriction_{[j_1,j_2]}$ by r' and repeat this case distinction on the resulting run again.

In the second case, we always choose a maximal j such that the topmost counter is 0. We then replace all occurring configurations by others that do not assume

[5] We use $\bigsqcup$ as the symbol for the disjoint union.

the counter value 0 on the topmost counter. Thus, if we iterate this process, the number j in each step strictly decreases. Since the run is finite, after some iterations, we must reach the contradiction to the first case.

Thus, we conclude that $\mathsf{ret}_k((\sigma, m)) = \mathsf{ret}_k((\sigma, \hat{m}))$ for all $k \in \mathbb{N}$ and all $m \geq \hat{m}$. This immediately implies the analogous result for $k = \infty$.

The claim for loops is proved completely analogous: there is a value m_∞ between $|\Sigma||Q|^2$ and $2 \cdot |\Sigma||Q|^2$ such that

$$\mathsf{lp}_\infty((\sigma, m_\infty)) = \mathsf{lp}_\infty((\sigma, m_\infty + 1)). \tag{1}$$

By a similar case distinction as in the return case, also from this point on the loops stabilise. The only difference now is that a counter value 0 can occur within a return starting with a topmost counter value $m_\infty + 1$ or within a loop starting with topmost counter value $m_\infty + 1$. Nevertheless either the first part of the lemma or (1) allow to replace this subrun by one not visiting configurations with topmost counter value 0. $\square$

Proof (of Lemma 10). Since for all $\sigma \in \Sigma$ and $i \in \mathbb{N}$ the sequence $\mathsf{ret}_k((\sigma, i))$ is monotone in k, there is a number $k_0 \leq |\Sigma|^2 \cdot |Q|^4$ such that $\mathsf{ret}_{k_0}((\sigma, i)) = \mathsf{ret}_{k_0+1}((\sigma, i))$ for all $1 \leq i \leq |\Sigma| \cdot |Q|^2$ and all $\sigma \in \Sigma$. Due to Lemma 9, we conclude that for all $i \in \mathbb{N}$ and all $\sigma \in \Sigma$, $\mathsf{ret}_{k_0}((\sigma, i)) = \mathsf{ret}_{k_0+1}((\sigma, i))$.

Similar to the previous proof we now show that $\mathsf{ret}_{k_0} = \mathsf{ret}_k$ for all $k \geq k_0$. For $k \leq k_0 + 1$, this is already guaranteed by choice of k_0. Assume that there are $\sigma \in \Sigma$, $k > k_0 + 1$ and $i \in \mathbb{N}$ such that $(q, q') \in \mathsf{ret}_k((\sigma, i)) \setminus \mathsf{ret}_{k_0}((\sigma, i))$ and that r is a return witnessing this fact. We assume that k is minimal whence

$$\mathsf{ret}_{k_0} = \mathsf{ret}_{k-1}. \tag{2}$$

Thus, r is a run that increases the height by k. It decomposes as $r = m_0 \circ p_0 \circ r_0 \circ m_1 \circ p_1 \circ r_1 \circ \cdots \circ m_i \circ p_i \circ r_i \circ m_{i+1} \circ s$ where each m_i is a subrun only using stay_f-operations (whence all configurations have the same height as the initial one), p_i is a subrun performing only one $\mathsf{push}_{\sigma_i, f_i}$, r_i is a return, and s is a subrun performing only one pop-operation. Now, some of the r_i increase the height by $k - 1$. By (2), we can replace each such r_i by some return r'_i that increases the height by at most k_0. This shows $(q, q') \in \mathsf{ret}_{k-1}((\sigma, i)) = \mathsf{ret}_{k_0}((\sigma, i))$ contradicting our assumption.

Thus, $\mathsf{ret}_k = \mathsf{ret}_{k_0}$ for all $k \geq k_0$ whence also $\mathsf{ret}_\infty = \bigcup_{k \in \mathbb{N}} \mathsf{ret}_k = \bigcup_{k \leq k_0} \mathsf{ret}_k = \mathsf{ret}_{|\Sigma|^2 \cdot |Q|^4}$.

The proof for $\mathsf{lp}_\infty = \mathsf{lp}_{|\Sigma|^2 \cdot |Q|^4 + 1}$ follows because whenever a loop increases the height of the stack, it continues with some return. By the result for returns, this subreturn can be replaced by one that only increases the height by $|\Sigma|^2 \cdot |Q|^4$. $\square$

B Reachability for k-HOCA$^-$ and k-HOCA$^+$

B.1 Auxiliary Storage Automata

Following Engelfriet's approach [7], we use *Auxiliary* $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ *automata* (where $b : \mathbb{N} \to \mathbb{N}$ is some function) for the analysis of $\mathcal{S}$ automata. The former are a general model for computing. An instance is given by

1. a finite control structure with control states Q,
2. an initial state $q_0 \in Q$,
3. transition rules Δ,
4. a two-way read-only input tape,
5. a worktape (like for Turing machines) of size $b(n)$ where n is the size of the input, and
6. a storage $\mathcal{S}$.

The storage $\mathcal{S}$ can be any known storage type used for automata, e.g., stacks, pushdowns, or counters. As usual, the above introduced automata can be deterministic, nondeterministic, or alternating. We refer to [7] for a detailed formal introduction, the connection to Turing-machine based notions of time and space complexity, as well as for references to the classical literature on these machine models.

Note, that a *1-way* auxiliary $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ automaton is defined analogously whereas the input tape is only read one-way. Most classical automata models can be directly rendered into this framework, e.g., nondeterministic 1-way $\mathcal{S}$ automata where $\mathcal{S}$ is the trivial storage correspond to nondeterministic finite automata; 2-way auxiliary $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ automata where $\mathcal{S}$ is the trivial storage are the classical $b(n)$-space bounded Turing machines; 1-way $\mathcal{S}$ automata where $\mathcal{S}$ is a pushdown correspond to pushdown automata.

The configuration of an auxiliary $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ automaton is the tuple containing the current finite state $q \in Q$, the contents of the auxiliary work tape, i.e., a word w of size bounded in $\mathbf{SPACE}(b(n))$, as well as the configuration $x \in X$ of $\mathcal{S}$. As usual, we define a run of an automaton as a sequence of configurations that is conform with the underlying transition rules and the semantics of the storage type. The applicable transition rules depend on the outcome of the storage tests applied to the current storage configuration, the current control state and the next input symbol.

B.2 Technical Results

Before we analyse control-state reachability on k-HOCA$^+$ and k-HOCA$^-$, we recall and extend some results of Engelfriet. The following results are Theorems 2.2 and 2.4 of [7].

Lemma 17. *Let b be some function satisfying $b(n) \geq \log(n)$ for all $n \in \mathbb{N}$ and let $\mathcal{S}$ be a storage type. In polynomial time, we can translate an alternating auxiliary $\mathbf{SPACE}(\bigcup_{d\in\mathbb{N}} \exp(db(n)))$ $\mathcal{S}$ automaton into an alternating auxiliary $\mathbf{SPACE}(b(n))$ $\mathcal{P}(\mathcal{S})$ automaton such that both automata accept the same language and vice versa.*

Lemma 18. *For $b(n) \geq \log(n)$ and every storage type $\mathcal{S}$, there are polynomial time algorithms that translate a nondeterministic auxiliary* $\mathbf{SPACE}(b(n))$ $\mathcal{P}(\mathcal{S})$ *automaton into an alternating auxiliary* $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ *automaton accepting the same language and vice versa.*

A detailed look on Engelfriet's proof of Lemma 7.11 in [7] reveals that its analogue for alternating automata holds if we replace the role of nonemptyness by the role of control state reachability. Moreover, the logspace reduction of membership for auxiliary $\mathbf{SPACE}(log(n))$ $\mathcal{S}$ automata to control state reachability for 1-way $\mathcal{S}$ automata extends to an $b(n)$ space-bounded reduction of membership for auxiliary $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ automata to control state reachability for $\mathcal{S}$ automata (which now may have size $\exp(b(n))$ when starting with an input of size n). Before we prove these claims, let us first define alternating reachability in our setting.

Definition 19. *Let $\mathcal{A}$ be an alternating auxiliary* $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ *automaton. For a set C of configurations of $\mathcal{A}$ we define $\mathsf{pre}^*(C)$ as the set of configurations c such that there is a computation tree T of $\mathcal{A}$*

1. *the root of T is labelled by c,*
2. *all leaves of T are labelled by configurations $c' \in C$,*
3. *for each inner node t of T labelled by an existential configuration c there is exactly one successor t' in T and t' is labelled by a successor c' of c (w.r.t $\mathcal{A}$), and*
4. *for each inner node t of T labelled by a universal configuration c there is, for each successor c' of c (w.r.t $\mathcal{A}$) a successor t' of t labelled by c'.*

For a state q of $\mathcal{A}$, we say that q is alternatingly (control state) reachable in $\mathcal{A}$ if the initial configuration of $\mathcal{A}$ belongs to $\mathsf{pre}^*(\{(q,x) \mid x \text{ is an } \mathcal{S}\text{-configuration}\})$.

Remark 20. Note that we disallow that the computation tree T may contain a leaf labelled by some universal state c not in C such that no transition of $\mathcal{A}$ is applicable to c. This restrictive definition is necessary for the results provided in the following.

Definition 21. *The alternating control state reachability problem for some class $\mathcal{C}$ of automata is the following.*
Input: $\mathcal{A} \in \mathcal{C}$, q a state of $\mathcal{A}$
Output: Is q alternatingly reachable in $\mathcal{A}$.

The following lemmas extend Engelfriet's result on the connection between emptiness (or equivalently, control state reachability) of nondeterministic $\mathcal{S}$ automata and membership for nondeterministic auxiliary $\mathbf{SPACE}(\log(n))$ $\mathcal{S}$ automata to the setting of alternating automata.

Lemma 22. *Let $\mathcal{S}$ be a storage type. Alternating (or nondeterministic, respectively) control state reachability of (1-way) $\mathcal{S}$ automata reduces to membership of alternating (nondeterministic, respectively) auxiliary* $\mathbf{SPACE}(\log(n))$ $\mathcal{S}$ *automata via logspace reductions.*

Proof. Let $A(\mathcal{S})$ denote the set of 1-way alternating $\mathcal{S}$ automata and fix an effective encoding $\overline{A}(\mathcal{S})$ of this set as binary strings. We write $\overline{\mathcal{M}} \in \bar{A}(\mathcal{S})$ for the encoding of the automaton $\mathcal{M} \in A(\mathcal{S})$. Analogously, we write $\bar{q}$ for the encoding of some state q. We define an auxiliary $\mathbf{SPACE}(\log(n))$ $\mathcal{S}$ automaton which we call $\mathcal{A}$ such that $L(\mathcal{A}) = \{\overline{\mathcal{M}}\#\bar{q} \mid \mathcal{M} \in A(\mathcal{S}), q$ a state of $\mathcal{M}$ and q alternating reachable by $\mathcal{M}\}$. Given an input string s, $\mathcal{A}$ first checks that $s = \overline{\mathcal{M}}\#\bar{q}$ for some $\mathcal{M} \in A(\mathcal{S})$ and some state q of $\mathcal{M}$. Now $\mathcal{A}$ simulates $\mathcal{M}$ storing two pointers on its tape, one called state pointer and one called transition pointer.

As initialisation the state pointer is set to the position of the input where the initial state of $\mathcal{M}$ is encoded.

We now iterate the following case distinction. If the state pointer points to (the encoding of) q, $\mathcal{A}$ accepts. Otherwise, the state pointer points to some state q'. Scanning the input string we determine whether q' is an existential state of $\mathcal{M}$. If this is the case, we do an existential simulation step, otherwise we do a universal simulation step.

– *Existential simulation step.* The state pointer points to some state q'. Now we guess a transition applicable to the current configuration of $\mathcal{M}$ (which is (q', x) for x the current storage configuration of $\mathcal{A}$. This is done by setting the transition pointer to some value i such that at position i in the input string the encoding of a transition $\delta = (p, t, f, r)$ starts. $\mathcal{A}$ now checks that $p = q'$. Then it checks that the test formula t is satisfied by the current storage configuration. If this is not the fact, $\mathcal{A}$ rejects. Otherwise it applies f to the storage and changes the state pointer such that it points to the encoding of r.
– *Universal simulation step.* The state pointer points to some universal state $q' \neq q$ and the current $\mathcal{S}$ configuration (of $\mathcal{A}$ and of $\mathcal{M}$) is x. Recall that q is alternatingly reachable from (q', x) if there is a computation tree where the root is labelled by (q', x) and is not a leaf (because $q \neq q'$) and q is alternatingly reachable from every successor of (q', x) in the computation tree.

  In order to guarantee that (q', x) has a successor configuration with respect to $\mathcal{M}$, we universally spawn a subprocess that performs an existential simulation step. If this branch accepts, we still have to show that for any applicable transition, q is alternatingly reachable from the resulting configuration. For this purpose the transition pointer iterates over all positions in the encoding $\overline{\mathcal{M}}$ of $\mathcal{M}$. As soon as this iteration has been finished, this main process accepts. During the iteration it may spawn subprocesses as follows.

  If the current position of the pointer points to a transition $\delta = (p, t, f, r)$ we check whether $q' = p$. In this case we universally spawn a subprocess. It checks whether the test formula t is satisfied by the current storage configuration. If not, the process accepts. Otherwise, $\mathcal{A}$ universally branches to an accepting branch and another branch by first applying f to the current storage configuration and then setting the state pointer to the position of the encoding of r and starting the next simulation step. It is straightforward to see that one of the following holds.

1. f is not applicable to the current storage configuration, thus δ does not provide a successor of the current configuration of $\mathcal{M}$. In this case, the universal branching only spawns one accepting branch whence the subprocess dealing with δ accepts.
2. f is applicable to the current storage configuration. Then the subprocess applying f accepts if and only if q is alternatingly reachable from the δ-successor of the current configuration.

It is straightforward to prove that $\mathcal{A}$ accepts $\overline{\mathcal{M}}\#\bar{q}$ if and only if q is alternatingly reachable by $\mathcal{M}$. Moreover, $\mathcal{A}$ only needs universal states for the universal simulation step. Thus, restricting the input to nondeterministic $\mathcal{S}$ automata, the simulating machine $\mathcal{A}$ will also be nondeterministic instead of alternating. $\square$

Engelfriet also provided a logspace reduction in the other direction in the nondeterministic case. We extend this result again to the alternating case and to auxiliary $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ automata for arbitrary space bound b.

Lemma 23. *Let $b(n) \geq \log(n)$ and $\mathcal{M}$ be an alternating (or nondeterministic) auxiliary $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ automaton. The membership problem for $\mathcal{M}$ is reducible to alternating (nondeterministic, respectively) control state reachability for $\mathcal{S}$ automata via a $\mathbf{DSPACE}(b(n))$-computation.*

Proof. Let Γ be the tape alphabet with blank symbol $\square$, Σ the input alphabet, Q the state set and $q_0 \in Q$ the initial state of $\mathcal{M}$. Without loss of generality $\mathcal{M}$ has only 1 accepting state and it enters this state if and only if the tape is completely blank and the heads of the input and the reading tape are on the first cell.

On input a word w, we construct an automaton $\mathcal{A}$ with state set $Q \times \Gamma^{b(|w|)} \times \{1, 2, \dots b(|w|)\} \times \{1, 2, \dots, |w|\}$. Note that each configuration fits into space $O(b(|w|))$. The initial state is $(q_0, w\square^{b(|w|)-|w|}, 1, 1)$. Some state $c = (q, \gamma_1 \dots \gamma_{b(|w|)}, i, j)$ represents the configuration of $\mathcal{M}$ where the work tape contains the letters $\gamma_1 \dots \gamma_{b(|w|)}$, $\mathcal{M}$ is in state q the head of the work tape is at position i and the head of the input tape is at position j. This state is an existential one if and only if q is an existential state of $\mathcal{M}$. $\mathcal{A}$ has a transition (c, t, f, c') to state $c' = (q', \gamma'_1 \dots, \gamma'_{b(|w|)}, i', j')$ if and only if $\mathcal{M}$ has a transition with test-formula t and storage operation f whose application would translate configuration c to configuration c' (for all storage configurations where t is satisfied and f is applicable). The final state of $\mathcal{A}$ is $c_f = (q_f, \square^{b(|w|)}, 1, 1)$.

It is straightforward to prove that c_f is alternatingly reachable by $\mathcal{A}$ if $\mathcal{M}$ accepts w. Note that $\mathcal{A}$ contains universal states if and only if $\mathcal{M}$ contains universal states. $\square$

Analogously to Engelfriet's proof that a pushdown can replace alternation, we now investigate tradeoffs concerning the storage type $\mathcal{Z}+$. This proof is inspired by the **PSPACE**-hardness proof for emptiness of alternating finite automata recently published by Jancar and Sawa [10].

Lemma 24. *Let $b(n) = \exp_k(n)$ for some $k \geq 0$. Let $\mathcal{M}$ be a deterministic auxiliary* $\mathbf{SPACE}(b(n))$ *automaton, i.e., a deterministic* $\mathbf{SPACE}(b(n))$ *Turing-machine. We can compute in logspace an alternating auxiliary* $\mathbf{SPACE}(\log(b(n)))$ *$\mathcal{Z}+$ automaton $\mathcal{A}$ such that $\mathcal{M}$ accepts w iff $\mathcal{A}$ accepts w for all $w \in \Sigma^*$.*

Proof. Assume that $\mathcal{M}$ has state set Q, initial state $q_0 \in Q$, final state $q_f \in Q$ and tape alphabet Γ. The main states of $\mathcal{A}$ come from the set $Z = \Gamma \cup (\Gamma \times Q)$. Moreover the state set Q' of $\mathcal{A}$ contains $p(|Z|)$ many auxiliary states for some polynomial p. For simplicity of the presentation we omit the formal specification of these states. Our goal is to construct an automaton $\mathcal{A}$ whose configurations are of the form $(z, t, i) \in Z \times \{0,1\}^{\log(b(n))} \times \mathbb{N}$ where z is the current state of $\mathcal{A}$, t is the content of its tape (which we identify with a binary encoded natural number between 0 and $b(n)$) and i is the current counter value.

Our goal is to define $\mathcal{A}$ in such a way that $\mathcal{A}$ accepts from configuration (z, t, i) on input w if at time step i of the computation of $\mathcal{M}$ at the t-th cell of $\mathcal{M}$'s tape, the content is z (where we say that the t-th cell content is $(q, \gamma) \in Q \times \Gamma$ if the cell contains γ and $\mathcal{M}$ is reading this cell in state q). Let $\mathsf{preds}(z)$ be the set of triples (z_1, z_2, z_3) such that the one-step computation of $\mathcal{M}$ on the tape described by $z_1 z_2 z_3$ leads to the replacement of z_2 by z. If $\mathcal{A}$ is in some configuration (z, t, i) with $0 < t < b(n)$ and $i > 0$, it nondeterministically chooses the hopefully correct triple $(z_1, z_2, z_3) \in \mathsf{preds}(z)$ and universally branches to configurations $(z_1, t-1, i-1), (z_2, t, i-1), (z_3, t+1, i-1)$. Note that a finite amount of auxiliary states suffices to calculate the tape content $t+1$ and $t-1$ from t. We now specify the acceptance condition. Configurations $(\square, 0, i)$ and $(\square, b(n), i)$ are accepting (for all $i \in \mathbb{N}$) while all other configurations with tape $t = 0$ or $t = b(n)$ are rejecting (again, only finitely many states are needed to check whether we are in one of these configurations). Assuming that the input is $w = a_1 \dots a_n$, let configuration $((q_0, a_1), 1, 0)$, configurations $(a_i, i, 0)$ for $2 \leq i \leq n$, and configurations $(\square, j, 0)$ for $j > n$ be all accepting. All other configurations with counter value 0 are rejecting. Note that this acceptance condition relies on the input and can be checked with finitely many auxiliary states. In a configuration $(z, t, 0)$ we parse the input word to the t-th letter and compare z to this letter (if w ended before, then z has to be the blank symbol $\square$).

An easy induction on i shows that there is an accepting computation of $\mathcal{A}$ starting in (z, t, i) with input w if and only if in the computation of $\mathcal{M}$ on w the t-th letter of the i-th configuration is z (where as before $z = (q, \gamma) \in Q \times \Gamma$ means that the content of the t-th cell is γ, $\mathcal{M}$'s head is positioned at the t-th cell and $\mathcal{M}$ is in state q).

Now we add to $\mathcal{A}$ an initialisation phase that, on input w, guesses a letter $\gamma \in \Gamma$, a number $t \leq b(|w|)$ and some number $i \in \mathbb{N}$ switching to configuration $((q_f, \gamma), t, i)$. Now $\mathcal{A}$ accepts w if and only if the computation of $\mathcal{M}$ on w is accepting. ◻

B.3 Control-State Reachability on k-HOCA$^+$

We prove the part of Theorem 14 on reachability. The claim for alternating reachability follows directly from this result as we will explain in Section B.5. We determine the exact complexity of reachability on k-HOCA$^+$. For the base case we use a result mentioned by Göller [8].

Lemma 25. *Alternating control state reachability for alternating (1-way) $\mathcal{Z}+$ automata is* **PSPACE***-complete.*

Proposition 26. *Control state reachability for* k-HOCA$^+$ *is* $\mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$*-complete for all $k \geq 2$.*

Proof. For containment, let us first consider the case $k = 2$. Given a 2-HOCA$^+$ $\mathcal{A}$ and a state q, control state reachability reduces by Lemma 22 to a membership problem for a (nondeterministic) auxiliary (2-way) $\mathbf{SPACE}(\log(n))$ $\mathcal{P}(\mathcal{Z}+)$ automaton. Due to Lemma 18 this automaton can be translated into an alternating auxiliary $\mathbf{SPACE}(\log(n))$ $\mathcal{Z}+$ automaton. Due to Lemma 23, membership for this machine is logspace reducible to alternating control state reachability on alternating (1-way) $\mathcal{Z}+$ automata which by Lemma 25 is solvable in $\mathbf{PSPACE} = \mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_0(n^d))$.

Now we proceed by induction on k. Given a k-HOCA$^+$ $\mathcal{A}$ ($k \geq 3$) and a state q, control state reachability reduces by Lemma 22 to a membership problem for a (nondeterministic) auxiliary (2-way) $\mathbf{SPACE}(\log(n))$ $\mathcal{P}^{k-1}(\mathcal{Z}+)$ automaton. Due to Lemma 18 this machine can be translated into an alternating auxiliary $\mathbf{SPACE}(\log(n))$ $\mathcal{P}^{k-2}(\mathcal{Z}+)$ automaton. We apply Lemma 17 and obtain an equivalent alternating auxiliary $\mathbf{SPACE}(n^d)$ $\mathcal{P}^{k-3}(\mathcal{Z}+)$ automaton (for some $d \in \mathbb{N}$). Again with Lemma 18 this is translated to a nondeterministic auxiliary $\mathbf{SPACE}(n^d)$ $\mathcal{P}^{k-2}(\mathcal{Z}+)$ automaton. Using the polynomial-space reduction from Lemma 23 we obtain a state q' and a (k − 1)-HOCA$^+$ $\mathcal{A}'$ of size $\exp(O(|\mathcal{A}|^d))$ such that q is reachable in $\mathcal{A}$ if and only if q' is reachable in $\mathcal{A}'$. By induction hypothesis the latter is decidable in space $\exp_{k-3}(|\mathcal{A}'|^{d'})$ for some $d' \in \mathbb{N}$. Thus, in terms of $|\mathcal{A}|$ the space is bounded by $\exp_{k-3}((\exp(O(|\mathcal{A}|)^d)^{d'})) = \exp_{k-2}(O(|\mathcal{A}|)^d)$. This completes the containment proof.

We now prove hardness. Recall that Lemma 24 provided a reduction of any membership problem in $\mathbf{DSPACE}(\exp_{k-2}(n^d))$ ($d \in \mathbb{N}$) to a membership problem for an alternating auxiliary $\mathbf{SPACE}(\exp_{k-3}(n^d))$ $\mathcal{Z}+$ automaton. Due to Lemma 17 this can be reduced to a membership problem for an alternating auxiliary $\mathbf{SPACE}(\log(n^d))$ $\mathcal{P}^{k-2}(\mathcal{Z}+)$ automaton. Furthermore, by Lemma 18 this reduces to a membership problem for a nondeterministic auxiliary $\mathbf{SPACE}(\log(n^d))$ $\mathcal{P}^{k-1}(\mathcal{Z}+)$ automaton. Finally, due to Lemma 23, there is a polynomial time reduction of this problem to a control state reachability problem for a (1-way) $\mathcal{P}^{k-1}(\mathcal{Z}+)$ automaton of size $O(n^d)$, i.e., reachability for k-HOCA$^+$. ◻

B.4 Control-State Reachability on $\mathsf{k\text{-}HOCA}^-$

Based on our result that control state reachability for $\mathsf{2\text{-}HOCA}^-$ is in $\mathbf{P}$ (Proposition 6), Engelfriet's machinery allows to determine the complexity of reachability in n-HOCA^- inductively. This proves the first half of Theorem 13. The claim on alternating reachability is proved in the following section.

Proposition 27. *For $k \geq 2$, the control state reachability problem for $\mathsf{k\text{-}HOCA}^-$ is in $\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$.*

Proof. We use Engelfriet's machinery and induction: the case reachability for $\mathsf{2\text{-}HOCA}^- \in \mathbf{P}$ has already been shown in Proposition 6. Given a $\mathsf{k\text{-}HOCA}^-$ $\mathcal{A}$ of level $k \geq 3$ and a state q, control state reachability reduces by Lemma 22 to a membership problem for some nondeterministic auxiliary $\mathbf{SPACE}(\log(n))$ $\mathcal{P}^{k-1}(\mathcal{Z})$ automaton. Due to Lemma 18 this automaton can be translated into an alternating auxiliary $\mathbf{SPACE}(\log(n))$ $\mathcal{P}^{k-2}(\mathcal{Z})$ automaton. We apply Lemma 17 and obtain an alternating auxiliary $\mathbf{SPACE}(dn)$ $\mathcal{P}^{k-3}(\mathcal{Z})$ automaton for some $d \in \mathbb{N}$. Again with Lemma 18 this is translated to a nondeterministic auxiliary $\mathbf{SPACE}(dn)$ $\mathcal{P}^{n-2}(\mathcal{Z})$ automaton. Finally, we use the polynomial-space reduction from Lemma 23 and obtain a state q' and a $(\mathsf{k}-1)$-HOCA^- $\mathcal{A}'$ of size exponential in that of $\mathcal{A}$ such that q is reachable in $\mathcal{A}$ if q' is reachable in $\mathcal{A}'$. By induction hypothesis, we can decide this in time $\exp_{n-3}(p'(|\mathcal{A}'|)) = \exp_{n-2}(p(|\mathcal{A}|))$ for some polynomials p and p'. ◻

B.5 Alternating Control State Reachability

We derive our results on alternating reachability by use of a much more general relation between the pushdown operator and alternation.

Proposition 28. *Given any storage type $\mathcal{S}$, alternating control state reachability for $\mathcal{S}$ automata is logspace reducible to control state reachability of $\mathcal{P}(\mathcal{S})$ automata and vice versa.*

Proof. Let $\mathcal{A}$ be an alternating $\mathcal{S}$ automaton and q some state. By Lemma 22 the alternating control state reachability problem for $(\mathcal{A}, q)$ reduces to a membership problem for an alternating auxiliary $\mathbf{SPACE}(\log(n))$ $\mathcal{S}$ automaton. This reduces by Lemma 18 to a membership problem for a nondeterministic auxiliary $\mathbf{SPACE}(\log(n))$ $\mathcal{P}(\mathcal{S})$ automaton. Finally, using Lemma 23 this problem reduces to a control state reachability problem for a nondeterministic $\mathcal{P}(\mathcal{S})$ automaton.

Using the 'nondeterminism' variant of Lemma 22, the other direction of Lemma 18 and the 'alternation' variant of Lemma 23, the control state reachability problem for $\mathcal{P}(\mathcal{S})$ automata similarly reduces to the alternating control state reachability problem for alternating $\mathcal{S}$ automata.

C Equivalence of Storages

In [17] the notion of a level k counter automaton with 0-test was defined differently from our notion of $\mathsf{k\text{-}HOCA}^+$ as follows. Basically Slaats uses the storage type $\mathcal{P}^{k-1}_{\{\perp\}}(\mathcal{Z}+)$ instead of $\mathcal{P}^{k-1}_{\{\perp,0,1\}}(\mathcal{Z}+)$. In the following we show that both variants lead to equivalent automata. Let us first recall the notion of equivalence of storage types (cf. [7]).

Definition 29. *Let $\mathcal{S}$ and $\mathcal{S}'$ be storages. $\mathcal{S}$ can* simulate *$\mathcal{S}'$, denoted as $\mathcal{S}' \preceq \mathcal{S}$, if for every one-way deterministic $\mathcal{S}'$ transducer there is a one-way deterministic $\mathcal{S}$ transducer defining the same transductions.*

$\mathcal{S}$ and $\mathcal{S}'$ are equivalent*, denoted as $\mathcal{S} \equiv \mathcal{S}'$, if $\mathcal{S} \preceq \mathcal{S}'$ and $\mathcal{S}' \preceq \mathcal{S}$.*

Remark 30. As pointed out by Engelfriet, for storage types $\mathcal{S}, \mathcal{S}'$ such that $\mathcal{S} \preceq \mathcal{S}'$, for $t \in \{$nondeterministic, alternating, deterministic$\}$ t $\mathcal{S}$ automata can be simulated by t $\mathcal{S}'$automata.

Recall that we defined the storage type $\mathcal{Z} = \mathcal{P}_{\{\perp\}}$. In the following, we also use $\mathcal{Z}$ as the operator $\mathcal{P}_{\{\perp\}}$ acting on other storage types. We call $\mathcal{Z}(\mathcal{S})$ the storage type *counter of $\mathcal{S}$*.

Proposition 31. *It holds that $\mathcal{Z}^{k-1}(\mathcal{Z}+) \equiv \mathcal{P}^{k-1}(\mathcal{Z}+)$.*

Proof. The direction from left to right is clear because $\mathcal{P}$ is an extension of $\mathcal{Z}$. We show how $\mathcal{Z}^{k-1}(\mathcal{Z}+)$ can simulate $\mathcal{P}^{k-1}(\mathcal{Z}+)$.

We first show that $\mathcal{S} := \mathcal{P}(\mathcal{Z}^{k-2}(\mathcal{Z}+))$ can be simulated by $\mathcal{S}' := \mathcal{Z}^{k-1}(\mathcal{Z}+)$. The idea is to encode the pushdown symbol of level k, by the level 1 counter value modulo 3 (recall that $\mathcal{P}$ uses the pushdown alphabet $\{\perp, 0, 1\}$). For this purpose we first replace in the $\mathcal{P}(\mathcal{Z}^{k-2}(\mathcal{Z}+))$ every $\mathsf{push}_\perp$ of level 1(i.e. a push applied to $\mathcal{Z}+$) by 3 $\mathsf{push}_\perp$ operations and each level 1 pop-operation by 3 pop-operations of level 1. This results in an equivalent $\mathcal{S}$ automaton where the level 1 counter value is always $0 \mod 3$. Next, without loss of generality we assume that the $\mathcal{S}$ automaton only uses instructions of the form $\mathsf{pop}, \mathsf{push}_{\sigma,\mathsf{id}}$ and stay_f. *For the rest of this simulation, we identify $\perp$ with the number* 2. We want to represent a pushdown symbol $\sigma \in \{0, 1, \perp\}$ by $\sigma \mod 3$ on the level 1 counter. We initialise $\mathcal{S}'$ by applying 2 $\mathsf{push}_\perp$ on level 1 (this results in the counter value 2, which is $2 \mod 3$ representing the initial symbol $\perp$. Now we simulate the operations on $\mathcal{S}$ by $\mathcal{S}'$-operations as follows (where we assume that the current $\mathcal{S}$-configuration x is simulated by $\mathcal{S}'$-configuration x'.

1. The top_γ test for $\gamma \in \{0, 1, \perp\}$ can be simulated as follows. apply $\mathsf{push}_{\perp,\mathsf{id}}$, then determine the topmost symbol $\gamma' \in \{0, 1, \perp\}$ by level 1 pop-operations (while the 0-test fails) determining the value of the topmost level 1 counter modulo 3. After finishing the test we restore the pushdown by a pop operation and just have to compare γ with γ'.
2. The *empty*? test on level 1 is simulated by first determining which top_γ test applies for $\gamma \in \{0, 1, \perp\}$ as in the simulation of top_γ. Then we perform γ

many pop-operations of level 1, then the *empty*? test of $\mathcal{S}'$ coincides with the *empty*? of $\mathcal{S}$. We restore the pushdown by γ many $\mathsf{push}_\perp$ operations of level 1.
3. A pop operation is simulated by pop.
4. A $\mathsf{push}_{\gamma,\mathsf{id}}$ operation is simulated by the following program: first determine the topmost symbol $\gamma' \in \{0, 1, \perp\}$ of x'. Then apply $\mathsf{push}_{\perp,\mathsf{id}}$, then apply γ' many level 1 pop-operations. No we apply γ many level 1 $\mathsf{push}_\perp$ operations.
5. A stay_f operations is simulated by stay_f if f is not an operation of level 1. If it is of level 1 we just duplicate it 3 times.

This completes the proof that $\mathcal{P}(\mathcal{Z}^{k-2}(\mathcal{Z}+))$ can be simulated by $\mathcal{Z}^{k-1}(\mathcal{Z}+)$. The lemma now follows by induction on k: we have shown that $\mathcal{P}^1(\mathcal{Z}+) \equiv \mathcal{Z}^1(\mathcal{Z}+)$. Assume that for some k we have $\mathcal{P}^{k-1}(\mathcal{Z}+) \equiv \mathcal{Z}^{k-1}(\mathcal{Z}+)$. By Theorem 1.3.1 of [7], we obtain

$$\mathcal{P}^k(\mathcal{Z}+) \equiv \mathcal{P}(\mathcal{P}^{k-1}(\mathcal{Z}+)) \equiv \mathcal{P}(\mathcal{Z}^{k-1}(\mathcal{Z}+)) \equiv \mathcal{Z}^k(\mathcal{Z}+).$$

□

Readers interested in a more throughout comparison of different possible definitions of higher-order one-couter automata are invited to have a look at Appendix F.

D Separation of Languages of Higher-Order Counter Automata With or Without 0-test

Under the assumption that

$$\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_k(n^d)) \subsetneq \mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_k(n^d)) \subsetneq \mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k+1}(n^d))$$

our results on the reachability problem for HOCA implies a strict separation of the languages of higher-order counters and higher-order pushdowns.

We first recall some results of Engelfriet that allows to shift results on 2-way auxiliary automata down to 1-way automata. We recall his proofs in order to extract the constructive content.

Let N-aux-$\mathbf{SPACE}(\log(n)) - \mathcal{S}$-L denote the languages accepted by nondeterministic auxiliary $\mathbf{SPACE}(\log(n))\mathcal{S}$ automata. Let T be the class of nondeterministic logspace transducers. Let $T^{-1}(\mathcal{L}) := \{\tau^{-1}(L) \mid \tau \in T, L \in \mathcal{L}\}$ be the class of languages obtained by application of transductions from T to languages from $\mathcal{L}$.

Recall that $\mathsf{VAL}(\mathcal{S})$ is the language of valid storage sequences for storage type $\mathcal{S}$. It is accepted by a deterministic $\mathcal{S}$ automaton with only 1 state q and no ε-transitions that works as follows. Transitions on input an $\mathcal{S}$-operation f are of the form $(q, f, \emptyset, f, q)$, i.e., $\mathcal{S}$ on input f applies f unconditionally and transitions on input a test $(t = r)$ are of the form $(q, (t = r), (t, r), \mathsf{id}_X, q) <$ i.e., computation continues if test t results in r and the storage remains unchanged.

Lemma 32 ([7], Lemma 7.1). *For every storage type $\mathcal{S}$, N-aux-$\mathbf{SPACE}(\log(n))-\mathcal{S}$-L $= T^{-1}(1N-\mathcal{S}) = T^{-1}(1D-\mathcal{S}) = T^{-1}(\{\mathsf{VAL}(\mathcal{S})\})$ where $1N-\mathcal{S}$ ($1D-\mathcal{S}$, respectively) denotes the class of languages accepted by nondeterministic (deterministic, respectively) $\mathcal{S}$-automata.*

Proof (sketch). Given a nondeterministic auxiliary $\mathbf{SPACE}(\log(n))-\mathcal{S}$ automaton $\mathcal{A}$ we can split it into two devices as follows. First we use a nondeterministic logspace transducer $\mathcal{T}$ that simulates $\mathcal{A}$ but instead of performing $\mathcal{S}$-tests or -operations it writes these on the output tape (tests are written together with the expected test result). Then we use the deterministic $\mathcal{S}$ automaton $\mathcal{S}$ recognising $\mathsf{VAL}(\mathcal{S})$ and check whether the output of $\mathcal{T}$ is a valid sequence of operations and tests of $\mathcal{S}$.

For the other direction, given a transducer and an $\mathcal{S}$-automaton, the language of their composition is recognised by a nondeterministic auxiliary $\mathbf{SPACE}(\log(n))-\mathcal{S}$ automaton which is a simple product of the two automata. □

A straightforward extension of Engelfriet's Corollary 7.2 from [7] is the following.

Corollary 33. *Let $\mathcal{S}$ and $\mathcal{S}'$ be storage types. If N-aux-$\mathbf{SPACE}(\log(n))-\mathcal{S}$-L $\not\subseteq$ N-aux-$\mathbf{SPACE}(\log(n)) - \mathcal{S}'$-L then $1D$-$\mathcal{S} \not\subseteq 1N$-$\mathcal{S}'$. In particular $\mathsf{VAL}(\mathcal{S}) \notin 1N$-$\mathcal{S}'$.*

Proof. Proof by contraposition: If $\mathsf{VAL}(\mathcal{S}) \in$ 1N-$\mathcal{S}'$ then N-aux-$\mathbf{SPACE}(\log(n))-\mathcal{S}$-L$= T^{-1}(\{\mathsf{VAL}(\mathcal{S})\}) \subseteq T^{-1}(\text{1N-}\mathcal{S}') =$ N-aux-$\mathbf{SPACE}(\log(n)) - \mathcal{S}'$-L □

Due to Lemmas 22 and 23, complexity results on control state reachability for $\mathcal{S}$-automata help to separate the classes N-aux-$\mathbf{SPACE}(\log(n)) - \mathcal{S}$-L for different storage types $\mathcal{S}$ as follows.

Lemma 34. *Let $\mathcal{S}$ be some storage type and $\mathcal{C}$ a complexity class closed under $\mathbf{DSPACE}(\log(n))$ reductions. If control state reachability for $\mathcal{S}$-automata is complete $\mathcal{C}$ (under $\mathbf{DSPACE}(\log(n))$-reductions), then N-aux-$\mathbf{SPACE}(\log(n))-\mathcal{S}$-L$= \mathcal{C}$.*

Proof. Assume that $\mathcal{A}$ is a nondeterministic auxiliary $\mathbf{SPACE}(\log(n))\mathcal{S}$ automaton accepting some language L. Then $L \in \mathcal{C}$ because Lemma 23 provides a $\mathbf{DSPACE}(\log(n))$-reduction from L to control state reachability for $\mathcal{S}$-automata which is in $\mathcal{C}$ by assumption. Thus, we conclude that N-aux-$\mathbf{SPACE}(\log(n))-\mathcal{S}$-L$\subseteq \mathcal{C}$.

Now let L be some language in $\mathcal{C}$. There is a $\mathbf{DSPACE}(\log(n))$-reduction φ such that for all words w, $\varphi(w)$ is an encoding of a nondeterministic $\mathcal{S}$-automaton $\mathcal{A}$ and a state q such that q is reachable in $\mathcal{A}$ if and only if $w \in L$. Due to Lemma 22, there is a $\mathbf{DSPACE}(\log(n))$-reduction ψ and a nondeterministic auxiliary $\mathbf{SPACE}(\log(n))\mathcal{S}$-automaton $\mathcal{A}'$ such that $\mathcal{A}'$ accepts $\psi(\varphi(w))$ if and only if q is reachable in $\mathcal{A}$ if and only if $w \in L$. Recall that logspace reducibility is a transitive relation because the i-th symbol of a logspace reduction can be recomputed on the fly in logspace. Using the very same trick, we can define a nondeterministic auxiliary $\mathbf{SPACE}(\log(n))\mathcal{S}$-automaton $\mathcal{A}''$ that, given the input w simulates a run of $\mathcal{A}'$ on $\psi(\varphi(w))$. Hence, $\mathcal{A}''$ accepts w if and only if $w \in L$. This shows that $L \in$ N-aux-$\mathbf{SPACE}(\log(n)) - \mathcal{S}$-L. □

The previous two lemmas directly imply Proposition 15. As a corollary of this proposition, our results on reachability for higher-order counters imply the language separations stated in Corollary 16. Moreover, if Proposition 15 separates the languages of $\mathcal{S}_1$-automata from those of $\mathcal{S}_2$-automata, then $\mathsf{VAL}(\mathcal{S}_2)$ is an example language that separates the two classes.

Proof (of Corollary 16). Containments are all trivial. Strict containment of $L((\mathsf{k}-1)\text{-}\mathsf{HOPA})$ in $L(\mathsf{k}\text{-}\mathsf{HOCA}^-)$ follows from the fact that we can recognise the language $\{a^n b^m \mid m \leq \exp_{k-1}(n)\}$ by a $\mathsf{k}\text{-}\mathsf{HOCA}^-$ (cf. [B]) but we cannot recognise it by a $(\mathsf{k}-1)\text{-}\mathsf{HOPA}$ (cf [5]).

Recall that

– the languages of auxiliary $\mathbf{SPACE}(\log(n))\mathcal{P}^{k-1}(\mathcal{P})$ are exactly those in $\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k-1}(n^d))$ (cf. [7]),
– the languages of auxiliary $\mathbf{SPACE}(\log(n))\mathcal{P}^{k-1}(\mathcal{Z}+)$ are exactly those in $\mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$ due to Theorem 14 and Lemmas 22 and 23,
– the languages of auxiliary $\mathbf{SPACE}(\log(n))\mathcal{P}^{k-1}(\mathcal{Z})$ are exactly those in $\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k-2}(n^d))$ due to Theorem 13 and Lemmas 22 and 23.

Application of the previous corollary to the inequation

$$\mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_k(n^d)) \subsetneq \mathbf{DSPACE}(\bigcup_{d\in\mathbb{N}} \exp_k(n^d)) \subsetneq \mathbf{DTIME}(\bigcup_{d\in\mathbb{N}} \exp_{k+1}(n^d))$$

yields $L(\mathsf{k}\text{-}\mathsf{HOCA}^-) \subsetneq L(\mathsf{k}\text{-}\mathsf{HOCA}^+) \subsetneq L(\mathsf{k}\text{-}\mathsf{HOPA})$. $\square$

Correspondingly, $\mathsf{VAL}(\mathcal{P}^{k+1})$ is a (collapsible) higher-order pushdown language of level $k+1$ recognised by a deterministic automaton with 1 state and no ε-transitions which is not recognised by any (collapsible) higher-order pushdown automaton of level k.

E Comparing Notions of Regularity

In this section, we compare the expressive power and succinctness of different notions of regularity for sets of configurations of $\mathcal{P}(\mathcal{Z})$ automata. Recall that we introduced in Section 3.2 a notion of regularity via the encoding in binary trees. From now on we write E-regularity for this notion.

E.1 2-Store Alternating Finite Automata

We will first compare E-regularity with the notion of regularity via 2-store alternating finite automata [2]. Since we introduce E only for $\mathcal{P}(\mathcal{Z})$ configurations, we restrict our presentation of 2-store automata also to this setting. Nevertheless the ideas presented here have straightforward extensions to the general setting of $\mathcal{P}(\mathcal{P})$ configurations.

Definition 35. *Let $\mathcal{A}'$ be a $\mathcal{P}(\mathcal{Z})$ automaton with state set Q'. An alternating 2-store automaton $\mathcal{A}$ (with respect to $\mathcal{A}'$) is an alternating automata $\mathcal{A} = (Q, \rho, F, \Sigma, \Delta)$ where $Q' \subseteq Q$ is a finite set of states, $\rho : Q \to \{\exists, \forall\}$ splits Q into existential and universal states, $F \subseteq Q$ the set of final states, $\Sigma \subseteq \{0, 1, \bot\} \times A$ a set of transition labels such that A is a* finite *set of alternating finite automata over input alphabet $\{\bot\}$, and $\Delta \subseteq Q \times \Sigma \times Q$*

An accepting computation of $\mathcal{A}$ on a $\mathcal{P}(\mathcal{Z})$ configuration is defined inductively. Let $x = x'(\tau, m)$ with $x' \in (\{0, 1, \bot\} \times \mathbb{N})^*$, $\tau \in \{0, 1, \bot\}$, and $m \in \mathbb{N}$, and let $q \in Q$ be a state. There is an accepting computation from q on x if one of the following holds.

1. *$x = \varepsilon$ and $q \in F$,*
2. *Assume that $x \neq \varepsilon$ and that $\rho(q) = \exists$. there is a $q' \in Q$ and a $(\tau, \mathcal{B})$ such that $(q, (\tau, \mathcal{B}), q') \in \Delta$, $\mathcal{B}$ accepts $\bot^m$, and there is an accepting computation from q' on x'.*
3. *Assume that $x \neq \varepsilon$ and that $\rho(q) = \forall$. For all $q' \in Q$ and a all $(\tau, \mathcal{B})$ such that $(q, (\tau, \mathcal{B}), q') \in \Delta$, $\mathcal{B}$ accepts $\bot^m$, and there is an accepting computation from q' on x'.*

For x a $\mathcal{P}(\mathcal{Z})$-configuration and $q \in Q$ a state of $\mathcal{A}'$, we say $\mathcal{A}$ accepts (q, x) if there is an accepting computation of $\mathcal{A}$ from q on x.

We call a set C of configurations of a $\mathcal{P}(\mathcal{Z})$ automaton 2-store-regular *if there is a 2-store automaton that accepts (q, x) if and only if $(q, x) \in C$.*

Remark 36. It is not difficult to adapt the usual powerset construction in order to obtain a deterministic 2-store automaton $\mathcal{A}'$ equivalent to a given alternating 2-store automaton. By deterministic, we mean that for any state q and any pushdown symbol $\tau \in \{0, 1, \bot\}$ there is exactly one deterministic automaton $\mathcal{B}$ and one state q' such that $(q, (\tau, \mathcal{B}), q')$ is a transition of $\mathcal{A}$. Of course this determinisation comes at the price of a blow-up of the state set.

Note that 2-store automata process the counter values stored in a $\mathcal{P}(\mathcal{Z})$ configuration sequentially. Thus, these automata cannot compare the values of different counters stored in the pushdown. To the contrary, in the tree-encoding of a configuration two adjacent counter values can be compared by just looking at the position where the two corresponding branches split up. Thus, we can define E-regular sets whose members satisfy certain restrictions with respect to the comparison of adjacent counter values. This idea can be translated into a proof that there is a E-regular set which is not 2-store-regular. After giving this proof, we show that 2-store-regular sets are always E-regular. These two results show that the expressive power of 2-store-regularity is strictly weaker than that of E-regularity.

Proposition 37. *There is a set C of configurations such that C is E-regular but not 2-store regular.*

Proof. Let $C = \{(q, (\bot, m)(\bot, m)) \mid m \in \mathbb{N}\}$.

C is clearly E-regular because $\mathsf{E}(C)$ contains a tree T if and only if there is some m such that the only leaves of T are 0^{m+1} and 0^m10. It is straightforward to design a tree-automaton for this set of trees.

Heading for a contradiction, assume that C is accepted by some alternating 2-store automaton $\mathcal{A}$. There are two numbers $m_0 \neq m_1$ such that the accepting runs of $\mathcal{A}$ on $(q, (\bot, m_0)(\bot, m_0))$ and $(q, (\bot, m_1)(\bot, m_1))$ use the same transitions of $\mathcal{A}$. In particular, both computations spawn the same alternating finite automata $\mathcal{A}_1, \dots, \mathcal{A}_m$ to accept $\bot^{m_0}$ or $\bot^{m_1}$, respectively. But then $\mathcal{A}$ also accepts $(q, (\bot, m_0)(\bot, m_1)) \notin C$ which is a contradiction.

Lemma 38. *Let C be a 2-store-regular set. Then C is E-regular.*

Proof. Let $\mathcal{A}$ be a 2-store automaton that recognises C. As explained in Remark 36, we may assume that $\mathcal{A}$ is deterministic. Let $\mathcal{B}_1, \dots, \mathcal{B}_n$ be the deterministic finite automata appearing in the transition labels of $\mathcal{A}$. Assume that $\mathcal{B}$ is the product automaton of $\mathcal{B}_1, \dots, \mathcal{B}_n$ and assume that the state sets of all $\mathcal{B}_i$ are pairwise disjoint. A tree-automaton accepting $\mathsf{E}(C)$ works as follows. It basically simulates all the $\mathcal{B}_k$ in parallel along all branches. Moreover, at every branching point of the tree it guesses the transition of $\mathcal{A}$ that connects the element of the pushdown encoded in the rightmost branch of the left subtree with the leftmost branch of the right subtree. The precise procedure is as follows.

Let $c := (q, (\tau_1, c_1) \dots (\tau_n, c_m))$. For each node d of $\mathsf{E}(c)$ the subtree of nodes comparable to d encodes some subpart $(q, (\tau_i, c_i) \dots (\tau_j, c_j))$ for $1 \leq i \leq j \leq m$. An accepting run on $\mathsf{E}(c)$ will label this node d with a tuple (q, p, r, s) where q, s are states of $\mathcal{A}$, p a state of $\mathcal{B}$ and r a state of some $\mathcal{B}_k$ $(1 \leq k \leq n)$ such that there is a run of $\mathcal{A}$ from state q to state s on $(\tau_i, c_i) \dots (\tau_j, c_j)$ such that the first transition of this run spawns a copy of $\mathcal{B}_k$ along the word $\bot^{c_i}$. This labelling is carried out in such a way that the labels of different nodes are compatible in the sense that the runs witnessed by the labels can be composed to an accepting run of $\mathcal{A}$ on c.

For this purpose, the left successor of the root is labelled by $L_0 := (q_0, p_0, r_0, s_0)$ where $q_0 = q$, s_0 is a final state of $\mathcal{A}$, p_0 is the initial state of some $\mathcal{B}_k$ and r_0 is the initial state of $\mathcal{B}$. Now the states are propagated as follows:

- If a node d with label $L_d = (q_d, p_d, r_d, s_d)$ has only a left successor (which is not a leaf, i.e., the tree label of $d0$ is $\bot$), then set $L_{d0} := (q_d, p_{d0}, r_{d0}, s_d)$ such that p_{d0} is the unique state such that $(p_d, \bot, p_{d0})$ is a transition of $\mathcal{B}_k$. Similarly r_{d0} is the successor of r_d with respect to $\mathcal{B}$.
- If a node d with label $L_d = (q_d, p_d, r_d, s_d)$ has a left successor (which is not a leaf, i.e., the tree label of $d0$ is $\bot$) and a right successor, then set $L_{d0} := (q_d, p_{d0}, r_{d0}, s_{d0})$ and $L_{d1} := (s_{d0}, p_{d1}, r_d, s_d)$ such that the following holds. p_{d0} is the unique state such that $(p_d, \bot, p_{d0})$ is a transition of $\mathcal{B}_k$. Similarly r_{d0} is the successor of r_d with respect to $\mathcal{B}$. s_{d0} is some state of $\mathcal{A}$ and p_{d1} is one of the components of r_d, i.e., a state of one of the $\mathcal{B}_{k'}$ as simulated by $\mathcal{B}$ up to this position.
- If the left successor of d is a leaf, and d's label is $L_d = (q_d, p_d, r_d, s_d)$ then we first compute L_{d0} and (if necessary) L_{d1} as in the steps before. If the

right successor exists, it is labelled by L_{d1}, the left successor is labelled by an accepting state if p_d is an accepting state of $\mathcal{B}_k$ such that $(q_d, (\tau, \mathcal{B}_k), s_{d0})$ is a transition of $\mathcal{A}$ where τ denotes the tree-label of the leaf at $d0$.

It is tedious but straightforward to prove that this tree-automaton accepts an encoding of a configuration if and only if it is in C. By taking a product with a tree-automaton recognising only valid encodings of configurations the claim is proved.

Unfortunately, the previous result that E-regularity is more expressive that 2-store-regularity does not imply that our result on the backwards or forward reachability carries over to 2-store-regular sets of configurations. The translation from the previous proof causes a blow-up of the state spaces. In the next lemma, we show that this blow-up is inevitable even if we start with deterministic 2-store automata.

Lemma 39. *There is a sequence $(\mathcal{A}_n)_{n\in\mathbb{N}}$ of deterministic 2-store automata such that there is no polynomial p and a sequence $(\mathcal{B}_n)_{n\in\mathbb{N}}$ of tree automata such that $|\mathcal{B}_n| \leq P(|\mathcal{A}_n|)$ and $\mathcal{B}_n$ accepts the same language as $\mathcal{A}_n$ (modulo translation with E.*

Proof. It is easy to design a deterministic 2-store multi-automaton $\mathcal{A}_n$ that accepts a configuration $(\bot, v_1)(\bot, v_2)\dots(\bot, v_m)$ if and only if

1. $m = n$, and
2. $v_i = 0 \mod p_i$ for all $1 \leq i \leq m$, where p_i denotes the i-th prime.

This is the automaton that goes from q_1 to q_2 to $\dots$ to q_{n+1} spawning in the i-th step an automaton checking the length of the input modulo p_i. This automaton can be realised with $n + 1 + \sum_{i=1}^{n} p_i \in O(n^3)$ states.

Assume that there is a $\mathcal{B}_n$ with less than 2^n many states accepting $\mathsf{E}(C)$ for C the configurations accepted by $\mathcal{A}_n$.

Set $m := \prod_{i=1}^{n} p_i$. There is an accepting run of $\mathcal{B}_n$ on $T := \mathsf{E}(q, (\bot, m)(\bot, m)\dots(\bot, m))$ where the height of the encoded pushdown is n. Note that $m > 2^n$ and the leaves of T are the nodes $0^{m+1}, 0^m10, 0^m1^20, \dots, 0^m1^{n-1}0$. Application of the pumping lemma for tree-automata yields that there is some $0 < k < m$ and a tree whose leaves are $0^{k+1}, 0^k10, 0^k1^20, \dots 0^k1^{n-1}0$ accepted by $\mathcal{B}_n$. But this tree encodes $(q, (\bot, k)(\bot, k)\dots(\bot k))$ where k is not divisible by all primes $p_1, \dots, p_m$. This contradicts the assumption that $\mathcal{B}_n$ accepts $\mathsf{E}(c)$ if and only if $\mathcal{A}_n$ accepts c.

E.2 Regularity via Sequences of Operations

Carayol [4] introduced a notion of regularity based on sequences of pushdown operations. He proved a normal form for this kind of regular sets which we present in the next definition. His notion also extends to higher-level pushdowns but for our purpose it suffices to restrict the presentation to sets of $\mathcal{P}(\mathcal{Z})$ configurations. In the following we write $\mathsf{Reg}(A)$ for the set of regular expressions over alphabet A and we write $L(r)$ for the languages $L \subseteq A^*$ defined by some regular expression $r \in \mathsf{Reg}(A)$.

Definition 40. *Let $t, s \in \mathsf{Reg}(\{\bot\})$ and $\sigma \in \{0, 1, \bot\}$. Then we define a binary relation $\xrightarrow{test(t)\sigma s}$ on $\Sigma \times \{\bot\}^*$ by $(\tau, \bot^k) \xrightarrow{t\sigma s} (\tau', \bot^{k'})$ if and only if $\bot^k \in L(t), \tau' = \sigma$, $k' \geq k$ and $\bot^{k'-k} \in L(s)$. We also define $(\tau, \bot^k) \xrightarrow{stest(t)\sigma} (\tau', \bot^{k'})$ if and only if $k' \leq k$, $\bot^{k-k'} \in L(s)$, $\bot^{k'} \in L(t)$ and $\tau' = \sigma$. These kind of definitions extend to expressions $e = \bigvee_{i=1}^{n_1} test(t_i)\sigma_i s_i \vee \bigvee_{i=1}^{n_2} r_i test(q_i)\tau_i$ via $\xrightarrow{e} := \bigcup_{i=1}^{n_1} \xrightarrow{test(t_i)\sigma_i s_i} \cup \bigcup_{i=1}^{n_1} \xrightarrow{r_i test(q_i)\tau_i}$. We write $\bigcup_e \xrightarrow{e}$ for the set of all such relations.*

A sequence regular expression *is an expression $e = \bigvee_{i=1}^{n} r_i s_i$ where each $r_i \in \mathsf{Reg}(\{\bot\})$ and each $s_i \in \mathsf{Reg}(\bigcup_e \xrightarrow{e})$.*

Each sequence-regular expression $e = \bigvee_{i=1}^{n} r_i s_i$ defines a set of configurations $L(e)$ as follows: $(\sigma_0, m_0)(\sigma_1, m_1) \dots (\sigma_n, m_n) \in L(e)$ if and only if $\sigma_0 = \bot$ and there is some $i \leq n$ such that $m_0 \in L(r_i)$ and there is a $w \in L(s_i)$ such that $w = \xrightarrow{e_0}\xrightarrow{e_1} \dots \xrightarrow{e_{n-1}}$ such that for all $j < n$ $(\sigma_j, m_j) \xrightarrow{e_j} (\sigma_{j+1}, m_{j+1})$.

We call a set C of configurations sequence-regular *if and only if there is a sequence-regular expression e such that $C = L(e)$.*

The main observation of this section is that the sets of sequence-regular sets are a strict subset of the set of tree-regular sets via E. For our proof we assume the reader to be familiar with pebble automata (cf. [C] for a survey). Moreover, we use the following results.

Lemma 41 ([C], Theorem 12 (cf. also [D])). *Positive cutting caterpillar expressions define the same tree languages as pebble automata.*

Lemma 42 ([E], Theorem 1.1). *The languages recognised by pebble automata are a strict subset of the languages recognised by tree-automata.*

Remark 43. We thank Mikołaj Bojańczyk for pointing out that the separating example can be easily adapted to be a set of trees T such that $T = \mathsf{E}(C)$ for a set of configurations C. Basically, one first translates the example into a set of unlabelled trees by encoding the labels as certain subtrees and then one adds the labels necessary to make the trees encodings of configurations.

Theorem 44. *The following holds:*

- *For each sequence-regular set C of configurations, C is E-regular.*
- *There is an E-regular set which is not sequence-regular.*

Proof. In fact, if P is sequence-regular, then $\mathsf{E}(P)$ is defined by a positive cutting caterpillar expression. This is due to the fact that the inorder traversal of $\mathsf{E}(c)$ for some configuration c visits the maximal paths in the order in which they appear as elements of the pushdown.

Assume that P is described by $r = \bigvee_{i=1}^{n} r_i s_i$. We translate r by structural induction into a (positive cutting) caterpillar expression r' recognising $\mathsf{E}(P)$. Since caterpillar expressions are closed under finite unions, it suffices to translate $r_i s_i$. Fix a $\mathcal{P}(\mathcal{Z})$ configuration $c = (\bot, m_0)(\sigma_1, m_1)(\sigma_2, m_2) \dots (\sigma_n, m_n)$ and let

$T = \mathsf{E}(c)$. In order to check that $c \in L(r_i s_i)$ we first have to check that $m_0 \in L(r_i)$. But this is equivalent to check that the leftmost branch of T is of the form $m_0\bot$. Thus, we modify r_i to r_i' by inserting a move to the left child before any letter occurring in r_i and add a final move to the left child and a check that the leaf is labelled by $\bot$.

Next we describe how to translate s_i into a caterpillar expression s_i' which leads to acceptance from the leftmost leaf of T if and only if $c \in L(r_i s_i)$ and the path to this leaf satisfies r_i'. Recall that s_i is a regular expression over relations $\stackrel{e}{\to}$. In order to satisfy s_i, we need to find a sequence of relations $\stackrel{e_i}{\to}$ such that $(\sigma_i, m_i) \stackrel{e_i}{\to} (\sigma_{i+1}, m_{i+1})$. Note that (σ_i, m_i) and (σ_{i+1}, m_{i+1}) are encoded by the paths to 2 adjacent leaves (in the inorder traversal). Thus, it suffices to gives a caterpillar expression $cp(e_i)$ that describes a pebble-automaton that runs from one leaf to the next leaf in the inorder traversal if and only if the corresponding paths are connected by $\stackrel{e_i}{\to}$. Once we have obtained such an expression, we can replace every occurrence of $\stackrel{e_i}{\to}$ by $cp(e_i)$ in s_i and composition of the resulting expression s_i' with r_i' has the property that $r_i' s_i'$ describes a pebble-automaton run from the root to the rightmost leaf on T if and only if $T = \mathsf{E}(c)$ for some $c \in L(r_i s_i)$ which completes the proof.

In order to obtain the expression $cp(e)$ for any relation $\stackrel{e}{\to}$ we make a case distinction on the form of e.

- Assume that $e = test(t)\sigma s$. In this case $cp(e)$ first uses the nesting operator in order to spawn a subexpression t' where every occurrence of $\bot$ in t is replaced by an arbitrary sequence of moves from a right successor to its parent and then one move from a left successor to a $\bot$ labelled parent. Afterwards, the main expression checks that we are at a leaf that is a left child, we move to the parent, then to the right child and then as in the translation of r_i we execute s along the leftmost branch of this subtree. Additionally, we check that the leaf of this leftmost branch is labelled by σ.
- Assume that $e = stest(t)\sigma$. In this case $cp(e)$ goes to the parent node until coming from a left child the node has a right child. Then it spawns a subexpression to the left child which evaluates s along the rightmost branch of this subtree. It also spawns a subexpression t' as in the previous case. Finally it goes to the right child and then to the left child. There it checks that this node is a leaf labelled σ.

Finally, when $cp(e)$ reaches the rightmost leaf, it accepts the whole tree.

Now using the expression $cp(e)$ instead of $\stackrel{e}{\to}$ in the s_i we can translate $r = \bigvee_{i=1}^{n} r_i s_i$ into $r' = \bigvee_{i=1}^{n} r_i' s_i'$ and obtain a positive cutting caterpillar expression that recognises $\mathsf{E}(P)$ for P the sequence-regular set we started with.

Remark 45. Similarly to our construction, it is easy to translate 2-store automata (after determinisation) into caterpillar expressions or sequence-regular expression. Thus, we have a strict hierarchy with respect to expressive power from 2-store-regularity via sequence-regularity to E-regularity.

As in the case of 2-store-regularity, sequence-regularity may provide more succinct descriptions of regular sets.

Lemma 46. *There is a sequence of sequence-regular expressions r_n of size polynomial in n such that there is no sequence of tree-automata $\mathcal{A}_n$ of size polynomial in n such that $\mathcal{A}_n$ recognises $\mathsf{E}(L(r_n))$ for each $n \in \mathbb{N}$.*

Proof. Let C be the set of configurations $(q, (\bot, c_1)(\bot, c_2)\dots(\bot, c_n))$ such that $c_1 = c_2 = \dots = c_n$ and c_i is divisible by the i-th prime. It is straightforward to write down a sequence-regular expression of polynomial size in n that describes C:

$r = rs$ where $r = (\bot^2)^*$ and $s = \xrightarrow{e_2}\xrightarrow{e_3}\dots\xrightarrow{e_n}$ where $\xrightarrow{e_i}$ is the identity function on all $(\bot, \bot^m)$ such that m is divisible by the i-th prime (which basically amounts to spawning the test $test((\bot^{p_i})^*)$).

To the contrary, as we have seen in the proof of Lemma 39, a tree-automaton recognising $\mathsf{E}(C)$ needs at least 2^n many states.

F Comparison of Expressive Power

In the last decades several equivalent definitions of higher-order pushdowns were used. Each of these can be restricted to unary stack alphabets resulting in a priori different kinds of storage types that could be called higher-order counters. In the following we show that most of these variants lead to storage types that lead to equivalent notions of nondeterministic higher-order counter automata. Note that our definition of higher-order counters leads to the most expressive variant of deterministic higher-order counter automata among those that we consider in the following.

First we consider higher-order pushdowns where only level 1 pushdowns contain stack symbols. By this we mean that a level k pushdown is not a list of pairs of stack symbols and level $k-1$ pushdowns but only a list of level $k-1$ pushdowns. We will show that this definition is equivalent to our definition in the case of higher-order pushdowns (which is well-known and straightforward) as well as for higher-order counters with 0-test. For higher-order counters without 0-test, we do not know whether the two versions are equivalent. At least it is clear that our version can simulate the more restricted version without higher-level pushdown symbols. Thus, our upper bounds, in particular the polynomial time algorithm for reachability on level 2, carry over to this setting. In fact, a simple adaptation of Slaat's proof [17] that k-HOCA^+ can simulate $(\mathsf{k}-1)$-HOPA shows that nondeterministic $(\mathsf{k}-1)$-HOPA can be simulated by this restricted version of nondeterministic k-HOCA^- the lower bounds also holds.

Finally, we also consider higher-order pushdown automata with inverse push-operations (cf. [5,A]). In these systems the level k pop-operation is replaced by a restricted version which is only applicable if the two topmost level $k-1$ pushdowns coincide. Carayol and Woehrle [5] have shown that this kind of higher-order pushdown storage is equivalent to the usual one for nondeterministic automata (see [A] for a proof). We show that this carries over to higher-order counters. In particular, even when we replace pop by the inverse push-operation and do not allow pushdown symbols on higher levels, the resulting nondeterministic higher-order counter automata with 0-test (without 0-test, respectively) is still equivalent to our notion of nondeterministic higher-order counter automata with 0-test (without 0-test, respectively). Some of these results carry over to deterministic automata as well. Before we go into the details we summarise our results in the following two theorems. We say $\mathcal{S}$ automata simulate $\mathcal{S}'$ automata if for each $\mathcal{S}'$ automaton there is a $\mathcal{S}$ automata recognising the same language and generating the same configuration graph after ε-contraction.

Theorem 47. *The following holds:*

1. *For any of the following storage types the nondeterministic r-way automata of one type can simulate the nondeterministic r-way automata of another type for ($r \in \{1,2\}$):*
   - $\mathcal{P}^k(\mathcal{P})$*, i.e., level k pushdown automata with pushdown symbols on each level (Engelfriet's definition of level k pushdown automata),*
   - $\mathcal{Z}^k(\mathcal{P})$*, i.e., level k pushdown automata with pushdown symbols only on level* 1 *(used for instance in [9]),*
   - $\mathcal{Z}^k_{inv}(\mathcal{P})$*, i.e., level k pushdown automata with pushdown symbols only on level* 1 *and inverse push operations (introduced in [5]),*
   - $\mathcal{P}^k_{inv}(\mathcal{P})$*, i.e., level k pushdown automata with pushdown symbols on each level and with inverse push operations.*
2. *The analogous statement for nondeterministic higher-order counter automata with* 0*-test also holds. For any of the following storage types the nondeterministic r-way automata of one type can simulate the nondeterministic r-way automata of another type (for $r \in \{1,2\}$):*
   - $\mathcal{P}^k(\mathcal{Z}+)$*, i.e., level k counter automata (with* 0*-test) with pushdown symbols on each level,*
   - $\mathcal{Z}^k(\mathcal{Z}+)$*, i.e., level k counter automata (with* 0*-test) with pushdown symbols only on level* 1*,*
   - $\mathcal{Z}^k_{inv}(\mathcal{Z}+)$*, i.e., level k counter automata (with* 0*-test) with pushdown symbols only on level* 1 *and inverse push operations,*
   - $\mathcal{P}^k_{inv}(\mathcal{Z}+)$*, i.e., level k counter automata (with* 0*-test) with pushdown symbols on each level and with inverse push operations.*
3. *For nondeterministic higher-order counter automata without* 0*-test we can only prove a weaker result. For any of the following storage types the nondeterministic r-way automata of one type can simulate the nondeterministic r-way automata of another type (for $r \in \{1,2\}$):*
   - $\mathcal{P}^k(\mathcal{Z})$*, i.e., level k counter automata (without* 0*-test) with pushdown symbols on each level,*
   - $\mathcal{Z}^k_{inv}(\mathcal{Z})$*, i.e., level k counter automata (without* 0*-test) with pushdown symbols only on level* 1 *and inverse push operations,*
   - $\mathcal{P}^k_{inv}(\mathcal{Z})$*, i.e., level k counter automata (without* 0*-test) with pushdown symbols on each level and with inverse push operations.*

   Moreover, nondeterministic $\mathcal{Z}^k(\mathcal{Z})$ automata can be simulated by each of the above mentioned automata.

Remark 48. All results of this theorem carry over to alternating automata analogously. Moreover, we can also add an auxiliary tape of size $\mathbf{SPACE}(b(n))$ for arbitrary function b.

Theorem 49. *The following holds:*

1. *For any of the following storage types the deterministic r-way automata of one type can simulate the deterministic r-way automata of another type (for $r \in \{1, 2\}$):*
   - $\mathcal{P}^k(\mathcal{P})$*, i.e., level k pushdown automata with pushdown symbols on each level,*
   - $\mathcal{Z}^k(\mathcal{P})$*, i.e., level k pushdown automata with pushdown symbols only on level* 1*,*
   - $\mathcal{P}^k_{inv}(\mathcal{P})$*, i.e., level k pushdown automata with pushdown symbols on each level and with inverse push operations.*

   Moreover deterministic $\mathcal{Z}^k_{inv}(\mathcal{P})$ automata can be simulated by any of the above mentioned automata types.
2. *The analogous statement for deterministic higher-order counter automata with* 0*-test also holds. For any of the following storage types the deterministic r-way automata of one type can simulate the deterministic r-way automata of another type (for $r \in \{1, 2\}$):*
   - $\mathcal{P}^k(\mathcal{Z}+)$*, i.e., level k counter automata (with* 0*-test) with pushdown symbols on each level,*
   - $\mathcal{Z}^k(\mathcal{Z}+)$*, i.e., level k counter automata (with* 0*-test) with pushdown symbols only on level* 1*,*
   - $\mathcal{P}^k_{inv}(\mathcal{Z}+)$*, i.e., level k counter automata (with* 0*-test) with pushdown symbols on each level and with inverse push operations.*

   Moreover deterministic $\mathcal{Z}^k_{inv}(\mathcal{Z}+)$ automata can be simulated by any of the above mentioned automata types.
3. *Deterministic $\mathcal{P}^k(\mathcal{Z})$ automata can simulate deterministic $\mathcal{P}^k_{inv}(\mathcal{Z})$ automata and vice versa. Moreover, deterministic $\mathcal{Z}^k_{inv}(\mathcal{Z})$ automata and deterministic $\mathcal{Z}^k(\mathcal{Z})$ automata are strictly weaker that $\mathcal{P}^k(\mathcal{Z})$ automata in the sense that every automaton of one of the former types can be simulated by some automaton of the latter type but not vice versa.*

Remark 50. The proof of this theorem will be based on Engelfriet's notion of equivalent storages. Thus, the statement remains valid, if we replace deterministic automata by any other kind of deterministic/nondeterministic/alternating r-way auxiliary $\mathbf{SPACE}(b(n))$ automata. It even carries over to the corresponding classes of transducers.

We conclude the presentation of the results of this section by pointing the reader to the open problems concerning equivalence of storage types.

Problem 51. 1. Is there some nondeterministic $\mathcal{P}^k(\mathcal{Z})$ automaton that cannot be simulated by any nondeterministic $\mathcal{Z}^k(\mathcal{Z})$ automaton?
2. Can we determinise the storage simulations that we so far only realised nondeterministic? In other words, can deterministic $\mathcal{Z}^k(\mathcal{S})$ automata be simulated by deterministic $\mathcal{Z}^k_{inv}(\mathcal{S})$ automata for $\mathcal{S}$ one storage type of the set $\{\mathcal{P}, \mathcal{Z}+, \mathcal{Z}\}$?

F.1 Simulation of Deterministic Automata

We first prove our claims about the deterministic case. Note that the nontrivial claims of Theorem 49 will be proved in Propositions 54 and 56, and in Corollary 62. Let us first recall the notion of equivalence of storage types (cf. [7]).

Definition 52. *Let $\mathcal{S}$ and $\mathcal{S}'$ be storages. $\mathcal{S}$ can* simulate *$\mathcal{S}'$, denoted as $\mathcal{S}' \preceq \mathcal{S}$, if for every one-way deterministic $\mathcal{S}'$ transducer there is a one-way deterministic $\mathcal{S}$ transducer defining the same transductions.*

$\mathcal{S}$ and $\mathcal{S}'$ are equivalent*, denoted as $\mathcal{S} \equiv \mathcal{S}'$, if $\mathcal{S} \preceq \mathcal{S}'$ and $\mathcal{S}' \preceq \mathcal{S}$.*

Remark 53. As pointed out by Engelfriet, this notion of equivalence implies that if $\mathcal{S} \preceq \mathcal{S}'$, then for $t \in \{\text{nondeterministic, alternating, deterministic}\}, r \in \{1, 2\}$, the t r-way auxiliary $\mathbf{SPACE}(b(n))$ $\mathcal{S}$ automata can be simulated by the t r-way auxiliary $\mathbf{SPACE}(b(n))$ $\mathcal{S}$' automata

Recall that we defined the storage type $\mathcal{Z} = \mathcal{P}_{\{\perp\}}$. In the following, we also use $\mathcal{Z}$ as the operator $\mathcal{P}_{\{\perp\}}$ acting on other storage types. We call $\mathcal{Z}(\mathcal{S})$ the storage type *counter of* $\mathcal{S}$. Apparently, the (first) $\perp$ component of every entry in the elements of $\mathcal{Z}(\mathcal{S})$ is redundant. Identifying $(\perp, c_1)(\perp, c_2)\dots(\perp, c_n)$ with $(c_1)(c_2)\dots(c_n)$ one sees easily that $\mathcal{Z}^{k-1}(\mathcal{P})$ automata are equivalent to the higher-order pushdown automata variant (of level k) used for instance in [9].

Proposition 54. *$\mathcal{Z}^k \preceq \mathcal{P}^{k-1}(\mathcal{Z})$, $\mathcal{Z}^{k-1}(\mathcal{Z}+) \equiv \mathcal{P}^{k-1}(\mathcal{Z}+)$ and $\mathcal{Z}^{k-1}(\mathcal{P}) \equiv \mathcal{P}^{k-1}(\mathcal{P})$.*

Proof. The direction from left to right is clear because $\mathcal{P}$ is an extension of $\mathcal{Z}$. We next show how $\mathcal{Z}^{k-1}(\mathcal{Z}+)$ can simulate $\mathcal{P}^{k-1}(\mathcal{Z}+)$.

We first show that $\mathcal{S} := \mathcal{P}(\mathcal{Z}^{k-2}(\mathcal{Z}+))$ can be simulated by $\mathcal{S}' := \mathcal{Z}^{k-1}(\mathcal{Z}+)$. The idea is to encode the pushdown symbol of level k, by the level 1 counter value modulo 3 (recall that $\mathcal{P}$ uses the pushdown alphabet $\{\perp, 0, 1\}$). For this purpose we first replace in the $\mathcal{P}(\mathcal{Z}^{k-2}(\mathcal{Z}+))$ every $\mathsf{push}_\perp$ of level 1(i.e. a push applied to $\mathcal{Z}+$) by 3 $\mathsf{push}_\perp$ operations and each level 1 pop-operation by 3 pop-operations of level 1. This results in an equivalent $\mathcal{S}$ automaton where the level 1 counter value is always $0 \mod 3$. Next, without loss of generality we assume that the $\mathcal{S}$ automaton only uses instructions of the form pop, $\mathsf{push}_{\sigma,\mathsf{id}}$ and stay_f. *For the rest of this simulation, we identify $\perp$ with the number* 2. We want to represent a pushdown symbol $\sigma \in \{0, 1, \perp\}$ by $\sigma \mod 3$ on the level 1 counter. We initialise $\mathcal{S}'$ by applying 2 $\mathsf{push}_\perp$ on level 1 (this results in the counter value 2, which is $2 \mod 3$ representing the initial symbol $\perp$. Now we simulate the operations on $\mathcal{S}$ by $\mathcal{S}'$-operations as follows (where we assume that the current $\mathcal{S}$-configuration x is simulated by $\mathcal{S}'$-configuration x'.

1. The top_γ test for $\gamma \in \{0, 1, \perp\}$ can be simulated as follows. apply $\mathsf{push}_{\perp,\mathsf{id}}$, then determine the topmost symbol $\gamma' \in \{0, 1, \perp\}$ by level 1 pop-operations (while the 0-test fails) determining the value of the topmost level 1 counter modulo 3. After finishing the test we restore the pushdown by a pop operation and just have to compare γ with γ'.

2. The *empty*? test on level 1 is simulated by first determining which top_γ test applies for $\gamma \in \{0, 1, \perp\}$ as in the simulation of top_γ. Then we perform γ many pop-operations of level 1, then the *empty*? test of $\mathcal{S}'$ coincides with the *empty*? of $\mathcal{S}$. We restore the pushdown by γ many $\mathsf{push}_\perp$ operations of level 1.
3. A pop operation is simulated by pop.
4. A $\mathsf{push}_{\gamma,\mathsf{id}}$ operation is simulated by the following program: first determine the topmost symbol $\gamma' \in \{0, 1, \perp\}$ of x'. Then apply $\mathsf{push}_{\perp,\mathsf{id}}$, then apply γ' many level 1 pop-operations. No we apply γ many level 1 $\mathsf{push}_\perp$ operations.
5. A stay_f operations is simulated by stay_f if f is not an operation of level 1. If it is of level 1 we just duplicate it 3 times.

This completes the proof that $\mathcal{P}(\mathcal{Z}^{k-2}(\mathcal{Z}+))$ can be simulated by $\mathcal{Z}^{k-1}(\mathcal{Z}+)$. The lemma now follows by induction on k: we have shown that $\mathcal{P}^1(\mathcal{Z}+) \equiv \mathcal{Z}^1(\mathcal{Z}+)$. Assume that for some k we have $\mathcal{P}^{k-1}(\mathcal{Z}+) \equiv \mathcal{Z}^{k-1}(\mathcal{Z}+)$. By Theorem 1.3.1 of [7], we obtain

$$\mathcal{P}^k(\mathcal{Z}+) \equiv \mathcal{P}(\mathcal{P}^{k-1}(\mathcal{Z}+)) \equiv \mathcal{P}(\mathcal{Z}^{k-1}(\mathcal{Z}+)) \equiv \mathcal{Z}^k(\mathcal{Z}+).$$

The equivalence $\mathcal{P}^k(\mathcal{P}) \equiv \mathcal{Z}^k(\mathcal{P})$ is obtained completely analogous.

We now want to discuss the variants of pushdown systems and counters with inverse push-operations. *For reasons of simplicity, we now consider the operator $\mathcal{P}$ to be restricted to $\mathsf{push}_{\sigma,\mathsf{id}}$, stay_f and pop operations.* Let $\mathcal{P}_{inv}$ and $\mathcal{Z}_{inv}$ be the variants of (the restricted) $\mathcal{P}$ and $\mathcal{Z}$ with inverse push-operations, i.e., $\mathcal{P}_{inv}$ is defined as $\mathcal{P}$ but instead of the operation pop we have the operation $\mathsf{push}^{-1}_{\gamma,\mathsf{id}}$. For $\mathcal{S}$ a storage type and $s := (\sigma_1, x_1) \dots (\sigma_{m-1}, x_{m-1})(\sigma_m, x_m)$ a $\mathcal{P}(\mathcal{S})$ configuration $\mathsf{push}^{-1}_{\gamma,\mathsf{id}}(s)$ is defined if and only if $\sigma_m = \gamma$ and $x_m = x_{m-1}$, i.e., if and only if $\mathsf{push}_{\gamma,\mathsf{id}}((\sigma_1, x_1) \dots (\sigma_{m-1}, x_{m-1})) = s$. In this case, $\mathsf{push}^{-1}_{\gamma,\mathsf{id}}(s) = \mathsf{pop}(s) = (\sigma_1, x_1) \dots (\sigma_{m-1}, x_{m-1})$.

Carayol and Woehrle[5] already showed that nondeterministic $\mathcal{Z}^k_{inv}(\mathcal{P})$ automata can simulate nondeterministic $\mathcal{Z}^k(\mathcal{P})$ automata and that $\mathcal{Z}^k_{inv}(\mathcal{P}) \preceq \mathcal{Z}^k(\mathcal{P})$. The latter simulation uses the fact that for every $\mathcal{Z}^k_{inv}(\mathcal{P})$-configuration there is a unique shortest sequence of operations that generates this configuration from the initial one. Moreover, a sequence s of operations translates one configuration $x_1$into another configuration x_2 if and only if the following holds. Let s_i be the unique sequence generating x_i, then s_2 results from $s_1 s$ by removing all adjacent pairs of inverse operations. Here, the inverse of $\mathsf{push}_{\gamma,\mathsf{id}}$ is $\mathsf{push}^{-1}_\gamma$ and the inverse of level 1 push_σ is pop_σ and the inverse of stay_f is $\mathsf{stay}_{f^{-1}}$ where f^{-1} is the inverse of f (pop_σ denotes a pop operation that is applied to a pushdown with topmost symbol σ). We next prove a similar result for $\mathcal{P}^k_{inv}(\mathcal{S})$ and $\mathcal{P}^k(\mathcal{S})$ for $\mathcal{S} \in \{\mathcal{P}, \mathcal{Z}, \mathcal{Z}+\}$ that even work deterministically in both directions.

Lemma 55. *For all $k \in \mathbb{N}$ $\mathcal{P}_{inv}(\mathcal{P}^k_{inv}(\mathcal{S})) \equiv \mathcal{P}(\mathcal{P}^k_{inv}(\mathcal{S}))$ for $\mathcal{S} \in \{\mathcal{P}, \mathcal{Z}, \mathcal{Z}+\}$.*

Proof. We first show $\mathcal{P}_{inv}(\mathcal{P}^k_{inv}(\mathcal{S})) \preceq \mathcal{P}(\mathcal{P}^k_{inv}(\mathcal{S}))$. This proof adapts the one of [A] and uses the level k symbols on the pushdown to store the minimal sequence that generated the current pushdown. For this purpose we replace the operations on $\mathcal{P}_{inv}(\mathcal{P}^k_{inv}(\mathcal{S}))$ as follows.

1. $\mathsf{push}_{\gamma,\mathsf{id}}$ is replaced by $\mathsf{push}_{(\gamma,\mathsf{push}_\gamma^{-1}),\mathsf{id}}$.
2. stay_f applied to a pushdown p represented by the pushdown p' is replaced by pop if the test $\mathsf{top}_{(\gamma,\mathsf{stay}_f)}(p') = true$ for some $\gamma \in \Gamma$. Otherwise, it is replaced by $\mathsf{push}_{(\gamma,\mathsf{stay}_{f^{-1}}),f}$ for γ such that $\mathsf{top}_\gamma(p) = true$.
3. $\mathsf{push}_\gamma^{-1}$ is replaced by pop if $\mathsf{top}_{(\gamma,\mathsf{push}_\gamma^{-1})}(p') = true$, otherwise it is undefined on p whence the simulation stops.

Adding some coding, one can translate the resulting system into one with topmost pushdown alphabet $\{\bot, 0, 1\}$. Correctness of this simulation follows from the results in [5,A].

For the other direction Carayol and Woehrle [5] proposed to simulate pop by guessing and creating the right level $k-1$ pushdown by push- and inverse push-operations of level below k and then apply an inverse level $(k-1)$-operation. This of course is a nondeterministic behaviour. Instead, we use their idea from the translation in the other direction: we annotate the pushdowns with the necessary operations in order to obtain the topmost pushdown of level $k-1$ for which the inverse push is applicable.

1. $\mathsf{push}_{\gamma,\mathsf{id}}$ is replaced by $\mathsf{push}_{(\gamma,\mathsf{pop}_\gamma),\mathsf{id}}$.
2. stay_f applied to a pushdown p represented by the pushdown p' is replaced by $\mathsf{stay}_f; \mathsf{push}^{-1}_{\gamma,\mathsf{stay}_f}$ if the test $\mathsf{top}_{(\gamma,\mathsf{stay}_f)}(p') = true$ for some $\gamma \in \Gamma$. Otherwise, it is replaced by $\mathsf{push}_{(\gamma,\mathsf{stay}_{f^{-1}}),f}$ for γ such that $\mathsf{top}_\gamma(p) = true$.
3. pop is replaced by a sequence performing $\mathsf{stay}_f; \mathsf{push}^{-1}_{\gamma,\mathsf{stay}_f}$ while the topmost level k symbol is $(\gamma, \mathsf{stay}_f)$. After iteration of this instruction, we end up with a topmost symbol (γ, pop) for some symbol γ. We then apply $\mathsf{push}^{-1}_{(\gamma,\mathsf{pop})}$.

Again using the usual coding trick, we can restrict the level k pushdown alphabet to $\{\bot, 0, 1\}$. The proof that this simulation is correct is completely analogous to the proof of the other direction.

This lemma allows to prove the following proposition:

Proposition 56. *$\mathcal{P}^k_{inv}(\mathcal{P}) \equiv \mathcal{P}^k(\mathcal{P})$, $\mathcal{P}^k_{inv}(\mathcal{Z}+) \equiv \mathcal{P}^k(\mathcal{Z}+)$, and $\mathcal{P}^k_{inv}(\mathcal{Z}) \equiv \mathcal{P}^k(\mathcal{Z})$,*

Proof. Let $\mathcal{S} \in \{\mathcal{P}, \mathcal{Z}+, \mathcal{Z}\}$. By induction on the previous lemma and the fact that the operator $\mathcal{P}$ preserves equivalence of storages (cf. [7]), we obtain

$$\mathcal{P}^k(\mathcal{S}) \equiv \mathcal{P}(\mathcal{P}^{k-1}_{inv}(\mathcal{S})) \equiv \mathcal{P}_{inv}(\mathcal{P}^{k-1}_{inv}(\mathcal{S}))) = \mathcal{P}^k_{inv}(\mathcal{S}).$$

We conclude this section by showing that $\mathcal{Z}^k(\mathcal{Z})$ is strictly weaker than $\mathcal{P}^k(\mathcal{Z})$ (and analogously for the variants with inverse push). In fact, we prove the stronger claim that any storage type with only trivial tests cannot deterministically simulate $\mathcal{Z}+$.

Definition 57. *Let $\mathcal{S} = (X, T, F, x_0)$ be a storage type. We call it* test-free *if the result of each test $t \in T$ is independent of the tested configuration, i.e., for all $t \in T$, and all $x, x' \in X$ we have $t(x) = t(x')$.*

Example 58. $\mathcal{Z}^k(\mathcal{Z})$ and $\mathcal{Z}^k_{inv}(\mathcal{Z})$ are test-free whereas $\mathcal{P}$ is not test-free.

In the following, we show that test-free storage types cannot deterministically compute any unbounded function f in the sense that the language $L_f := \left\{a^n b^{f(n)} \mid n \in \mathbb{N}\right\}$ is not recognised by any deterministic $\mathcal{S}$ automaton where $\mathcal{S}$ is a test-free storage type. In particular, test-free deterministic $\mathcal{S}$ automata do not accept $\{a^n b^n \mid n \in \mathbb{N}\}$ whence $\mathcal{Z}+ \not\preceq \mathcal{S}$. The crucial observation is that the storage configuration has no influence on the next transition except for the fact that it can abort a computation.

Lemma 59. *Let $\mathcal{S}$ be a test-free storage type and $\mathcal{A}$ a deterministic $\mathcal{S}$ automaton. For each input letter σ and all states q, the set of storage-configurations X splits into two disjoint sets $X = X_b \sqcup X_t$ such that*

- *for all configurations (q, x) with $x \in X_b$ no transition is applicable to (q, x), and*
- *there is a unique state p and a unique $\mathcal{S}$-operation o such that the unique successor configuration on reading ε or σ for each (q, x) with $x \in X_t$ is $(p, o(x))$.*

By induction on the length of a run we obtain the following corollary.

Corollary 60. *Let $\mathcal{S}$ be a test-free storage type and $\mathcal{A}$ a deterministic $\mathcal{S}$ automaton. For each state q there is a unique state p and a $\mathcal{S}$-operation o such that for each configuration (q, x) that admits a run on σ^k, the unique successor configuration after reading σ or ε is $(p, o(x))$. In particular, if (q, x) and (q, x') both allow a run reading σ^k these runs both end in the same state p'.*

Proposition 61. *Let $f : \mathbb{N} \to \mathbb{N}$ be an unbounded function and $\mathcal{S}$ a test-free storage type. $L_f = \left\{a^n b^{f(n)} \mid n \in \mathbb{N}\right\}$ is not recognised by any deterministic $\mathcal{S}$ automaton.*

Proof. Since f is unbounded, there is a state q and numbers $n_1, n_2 \in \mathbb{N}$ with $f(n_1) < f(n_2)$ such that the run on a^{n_i} ends in (q, x_i) for storage configurations x_1, x_2 of $\mathcal{S}$. By assumption there is a run from (q, x_1) reading $b^{f(n_1)}$ and ending in an accepting state p. Since (q, x_2) admits a run reading $b^{f(n_2)}$, it also a admits a run reading $b^{f(n_1)}$. Due to the previous corollary, this run ends in state p, whence $a^{n_2} b^{f(n_1)}$ is accepted. But this contradicts the fact that $a^{n_2} b^{f(n_1)} \notin L_f$ because $f(n_1) < f(n_2)$.

Corollary 62. *$\mathcal{Z}+ \not\preceq \mathcal{S}$ for any test-free storage type $\mathcal{S}$. In particular, $\mathcal{Z}+ \not\preceq \mathcal{Z}^k(\mathcal{Z})$ and $\mathcal{Z}+ \not\preceq \mathcal{Z}^k_{inv}(\mathcal{Z})$ for all $k \in \mathbb{N}$.*

Proof. There is a deterministic $\mathcal{Z}+$ automaton recognising $L_{\mathsf{id}} = \{a^n b^n \mid n \in \mathbb{N}\}$ which (by the previous proposition) is not recognised by any deterministic $\mathcal{S}$ automaton $\mathcal{A}$.

Since obviously $\mathcal{Z}+ \preceq \mathcal{P}^k(\mathcal{Z})$ for all $k \geq 1$, $\mathcal{P}^k(\mathcal{Z})$ is not equivalent to $\mathcal{Z}^k(\mathcal{Z})$ or $\mathcal{Z}^k_{inv}(\mathcal{Z})$.

Corollary 63. *$\mathcal{Z}^k(\mathcal{Z}) \prec \mathcal{P}^k(\mathcal{Z})$ and $\mathcal{Z}^k_{inv}(\mathcal{Z}) \prec \mathcal{P}^k(\mathcal{Z})$ for all $k \geq 1$.*

F.2 Simulation of nondeterministic automata

We now define a 'nondeterministic' version of the notion of equivalence of storage types. This allows to prove those parts of Theorem 47 that are not already implied by the results from the previous section.

Definition 64. *Let $\mathcal{S} = (X, T, F, x_0)$ and $\mathcal{S}' = (X', T', F', x'_0)$ be storage types. We say $\mathcal{S}$ can be nondeterministically simulated by $\mathcal{S}'$ and write $\mathcal{S} \preceq_N \mathcal{S}'$ if there is a map $\varphi : X \to X'$ such that the following holds.*

1. *There is a sequence $f_1, f_2, \dots, f_n \in F'$ such that $\varphi(x_0) = f_1(f_2(\dots f_n(x'_0)\dots))$.*
2. *For each $f \in F$ there is a nondeterministic $\mathcal{S}'$ automaton $\mathcal{A}_f$ with initial state q_i and final state q_f such that for all $x \in X$ there is a run of $\mathcal{A}_f$ from $(q_i, \varphi(x))$ to (q_f, y') if and only if $f(x)$ is defined and $y' = \varphi(f(x))$.*
3. *For each $t \in T$ there are two nondeterministic $\mathcal{S}'$ automaton $\mathcal{A}_t, \bar{\mathcal{A}}_t$ with initial states q_i and $\bar{q}_i$, and final states q_f and $\bar{q}_f$, respectively, such that*
   - *for all $x \in X$ there is a run of $\mathcal{A}_t$ from $(q_i, \varphi(x))$ to (q_f, y') if and only if $t(x) = true$ and $y' = \varphi(x)$, and*
   - *for all $x \in X$ there is a run of $\bar{\mathcal{A}}_t$ from $(\bar{q}_i, \varphi(x))$ to $(\bar{q}_f, y')$ if and only if $t(x) = false$ and $y' = \varphi(x)$.*

As in the case of $\preceq$, we write $\mathcal{S} \equiv_N \mathcal{S}'$ if $\mathcal{S} \preceq_N \mathcal{S}'$ and $\mathcal{S}' \preceq_N \mathcal{S}$.

Proposition 65. *Let $\mathcal{S} \preceq_N \mathcal{S}'$, $b(n) : \mathbb{N} \to \mathbb{N}$, $r \in \{1, 2\}$ and $t \in \{nondeterministic,\ alternating\}$. Every r-way t auxiliary* **SPACE**$(b(n))$ *$\mathcal{S}$ automaton $\mathcal{A}$ is simulated by some r-way t auxiliary* **SPACE**$(b(n))$ *$\mathcal{S}$' automaton $\mathcal{A}'$ in the sense that the configuration graphs of $\mathcal{A}$ and $\mathcal{A}'$ coincide after ε-contraction and both automata accept the same language.*

Proof. By a straightforward product construction of $\mathcal{A}$ and the $(\mathcal{A}_f)_{f \in F}$, $(\mathcal{A}_t)_{t \in T}$ and $(\bar{\mathcal{A}}_t)_{t \in T}$. Instead of executing $\mathcal{S}$-tests or -operations the automaton guesses the correct test result and then checks its guess and simulates the operation by executing first the corresponding $\mathcal{A}_t/\bar{\mathcal{A}}_t$ and the the corresponding $\mathcal{A}_f$ from its initial to its final state.

As in the deterministic case, the pushdown operator is monotone with respect to $\preceq_N$.

Proposition 66. *Let $\mathcal{S} \preceq_N \mathcal{S}'$. We have $\mathcal{P}(\mathcal{S}) \preceq_N \mathcal{P}(\mathcal{S}')$, $\mathcal{P}_{inv}(\mathcal{S}) \preceq_N \mathcal{P}_{inv}(\mathcal{S}')$, $\mathcal{Z}(\mathcal{S}) \preceq_N \mathcal{Z}(\mathcal{S}')$, $\mathcal{Z}_{inv}(\mathcal{S}) \preceq_N \mathcal{Z}_{inv}(\mathcal{S}')$,*

Proof. It suffices to provide simulations of the test $test(t)$ for each test t of $\mathcal{S}$ and simulations for the operations stay_f (note that $\mathsf{push}_{\gamma,f}$ can be replaced by $\mathsf{push}_{\gamma,\mathsf{id}}; \mathsf{stay}_f$).

The automaton $\mathcal{A}_{test(t)}$ that checks that $test(t) = true$ is equal to $\mathcal{A}_t$ but executes test $test(t')$ whenever $\mathcal{A}_t$ executes t' and performs $\mathsf{stay}_{f'}$ whenever $\mathcal{A}_t$ performs $\mathcal{S}'$-operation f'. Analogously we define $\bar{\mathcal{A}}_{test(t)}$ and $\mathcal{A}_{\mathsf{stay}_f}$.

Note that all storage types $\mathcal{S} = (X, T, F, x_0)$ we consider are *strongly connected* in the sense that for any $x, x' \in X$ there is a sequence $f_1, f_2, \ldots, f_n$ of $\mathcal{S}$-operations such that $x' = f_1(f_2(\ldots f_n(x)\ldots))$. As Carayol and Woehrle already noticed, pop of $\mathcal{P}(\mathcal{S})$ can be simulated nondeterministically by inverse push of $\mathcal{P}_{inv}(\mathcal{S})$ if $\mathcal{S}$ is strongly connected by simply guessing the right $\mathcal{S}$ configuration and restoring it before simulating the pop by an inverse push.

Lemma 67. *For strongly connected storage types $\mathcal{S}$, $\mathcal{Z}(\mathcal{S}) \preceq_N \mathcal{Z}_{inv}(\mathcal{S})$. Moreover, these storage types are again strongly connected.*

Induction on the previous lemma directly yields the following proposition.

Proposition 68. *For $k \in \mathbb{N}$ and $\mathcal{S} \in \{\mathcal{P}, \mathcal{Z}+, \mathcal{Z}\}$ we have $\mathcal{Z}^k(\mathcal{S}) \preceq_N \mathcal{Z}^k_{inv}(\mathcal{S})$.*

Proof. Inductively, $\mathcal{Z}^k(\mathcal{S}) \preceq_N \mathcal{Z}(\mathcal{Z}^{k-1}_{inv}(\mathcal{S})) \preceq_N \mathcal{Z}_{inv}(\mathcal{Z}^{k-1}_{inv}(\mathcal{S})) = \mathcal{Z}^k_{inv}(\mathcal{S})$.

The last claim we have to prove is that $\mathcal{Z}^k_{inv}(\mathcal{Z}) \equiv_N \mathcal{P}^k_{inv}(\mathcal{Z})$. Again we first prepare the proof by induction with a simple lemma.

Lemma 69. $\mathcal{P}_{inv}(\mathcal{Z}^{k-1}_{inv}(\mathcal{Z})) \equiv_N \mathcal{Z}_{inv}(\mathcal{Z}^{k-1}_{inv}(\mathcal{Z}))$ *for all $k \geq 1$.*

Proof. The $\succeq_N$ direction is trivial. For the other direction, we first do the case $k = 1$ and then the case $k \geq 2$.

A $\mathcal{P}_{inv}(\mathcal{Z})$ configuration $(\sigma_1, m_1)\ldots(\sigma_n, m_n)$ is identified with the $\mathcal{Z}_{inv}(\mathcal{Z})$ configuration $(\bot, m_1 + \sigma_1)(\bot, m_1)(\bot, m_2 + \sigma_2)(\bot, m_2)\ldots(\bot, m_n + \sigma_n)(\bot, m_n)$ (where we again identify $\bot$ with 2). In this representation a test for the topmost symbol is simple: the topmost symbol is σ if we can apply $\mathsf{stay}_{\mathsf{push}_\bot}$ σ many times follows by inverse push, push and σ many $\mathsf{stay}_{\mathsf{pop}}$ operations. The corresponding negative test is by guessing the symbol $\tau \in \{\bot, 0, 1\} \setminus \{\sigma\}$ and applying the positive test for τ. With the ability to test for the encoded topmost symbol, it is then easy to simulate any of the $\mathcal{P}_{inv}(\mathcal{Z})$ operations.

For the case $k \geq 3$ we use basically the same idea but we have to take care that we only encode one topmost symbol in the topmost level $k-1$ counter. For this purpose we define an auxiliary notation let $\sigma \in \{0, 1, \bot\}$, and m a $\mathcal{Z}^{k-1}_{inv}(\mathcal{Z})$ configuration. We write $m + \sigma$ for the result of applying to m the level 2 operation $\mathsf{push}_{\bot,\mathsf{id}}$ followed by the level 1 $\mathsf{push}_\bot$ for σ many times (level n means that we put the mentioned operation into a $(k-n)$-fold application of stay). We then encode a $\mathcal{P}_{inv}(\mathcal{Z}^{k-1}_{inv}(\mathcal{Z}))$ configuration $(\sigma_1, m_1)\ldots(\sigma_n, m_n)$ as the $\mathcal{Z}^k_{inv}(\mathcal{Z})$ configuration $(\bot, m_1 + \sigma_1)\ldots(\bot, m_n + \sigma_n)$. Simulation is now carried out similar to the case $k = 2$. The simulation of the test top_σ is by doing the right number of level 1 pop operations followed by a inverse push of level 2 and then again restoring the initial storage configuration. If we want to apply a storage operation (different from $\mathsf{push}_{\sigma,\mathsf{id}}$ and $\mathsf{push}^{-1}_\sigma$) to $(\bot, m_1 + \sigma_1)\ldots(\bot, m_n + \sigma_n)$, we first restore the configuration $(\bot, m_1 + \sigma_1)\ldots(\bot, m_n)$ then apply the configuration and afterwards restore the encoding of σ_n. For the $\mathsf{push}_{\sigma,\mathsf{id}}$ we just apply $\mathsf{push}_{\bot,\mathsf{id}}$ and subsequently replace the topmost $m_n + \sigma_n$ by $m_n + \sigma$. For the inverse push, we first have to guess σ_{n-1}, replace $m_n + \sigma_n$ by $m_n + \sigma_{n-1}$ and then apply the inverse push.

Again it is straightforward to prove that this simulation is correct.

Corollary 70. $\mathcal{Z}^k_{inv}(\mathcal{Z}) \equiv_N \mathcal{P}^k_{inv}(\mathcal{Z})$.

Proof. By induction on k and Lemmas 69 and Proposition 66, we obtain

$$\begin{aligned}\mathcal{P}^k_{inv}(\mathcal{Z}) &= \mathcal{P}_{inv}(\mathcal{P}^{k-1}_{inv}(\mathcal{Z})) \equiv_N \mathcal{P}_{inv}(\mathcal{Z}^{k-1}_{inv}(\mathcal{Z})) \\ &\equiv_N \mathcal{Z}_{inv}(\mathcal{Z}^{k-1}_{inv}(\mathcal{Z})) \equiv_N \mathcal{Z}^k_{inv}(\mathcal{Z})\end{aligned}$$

References

A. S. Wöhrle, "Decision problems over infinite graphs: Higher-order pushdown systems and synchronized products," Ph.D. dissertation, RWTH Aachen, 2005.

B. A. Blumensath, "On the structure of graphs in the Caucal hierarchy," *Theoretical Computer Science*, vol. 400, pp. 19–45, 2008.

C. M. Bojanczyk, "Tree-walking automata," tutorial at LATA 2007. Available: http://www.mimuw.edu.pl/~bojan/papers/twasurvey.pdf

D. B. ten Cate and L. Segoufin, "Xpath, transitive closure logic, and nested tree walking automata," in *Proc. of PODS'08*. ACM, 2008, pp. 251–260.

E. M. Bojanczyk, M. Samuelides, T. Schwentick, and L. Segoufin, "Expressive power of pebble automata," in *Proc. of ICALP'06*, LNCS 4051. Springer, 2006, pp. 157–168.