Optimal 2DFA Algorithms for One-Way Liveness on Two and Three Symbols

Christos A. Kapoutsis(✉)

Carnegie Mellon University in Qatar, Doha, Qatar
cak@cmu.edu

Abstract. For all $h, n \geq 2$, the problem of *one-way liveness* for height h and length n captures the task of simulating *one-way nondeterministic finite automata* with h states on inputs with $n-2$ symbols. We prove that the number of states in a smallest *two-way deterministic finite automaton* which decides this problem is $\Theta(h)$, if $n = 2$; and $\Theta(h^2/\log h)$, if $n = 3$.

1 Introduction

A long-standing open question at the intersection of automata theory and computational complexity is whether every *two-way nondeterministic finite automaton* (2NFA) with s states can be simulated by a *two-way deterministic finite automaton* (2DFA) with a number of states which is only polynomial in s. The well-known *Sakoda-Sipser conjecture* proposes that the answer is negative [5].

A stronger variant of this conjecture claims that, indeed, a 2DFA needs superpolynomially many states even when the input head of the 2NFA can only move forward, namely even when the 2NFA is really a *one-way nondeterministic finite automaton*. By [5], we know that this stronger claim holds iff a certain graph-theoretic problem called *one-way liveness* for height h (OWL_h) cannot be solved by any 2DFA whose number of states is only polynomial in h.

In one of the many approaches to this question, Hromkovič et al. [3] proposed the concept of a *reasonable automaton* (RA). Intuitively, this is a 2DFA with the following modifications: it works only on inputs of a fixed length n; it is *random-access*, in the sense that its head can move from any cell of its n-long input to any other in a single step; and, most importantly, it has every state associated with a propositional formula which indicates what the automaton 'knows' about its input when it is in that state. In this model, one uses the additional information provided by the formulas to study the different types of propositional reasoning that a 2DFA could possibly employ in trying to solve OWL_h on inputs of length n. Clearly, the standard 2DFA model would be too lean to support such a study.

Not surprisingly, the power of a RA varies as we vary (i) the set of propositional variables that can appear in the formulas and (ii) the rules for how these variables can be combined to create the formulas. By [3, Theorem 2], we know that a sufficiently expressive set of variables allows a RA to simulate any s-state 2DFA on

H.-J. Böckenhauer et al. (Eds.): Hromkovič Festschrift, LNCS 11011, pp. 33–48, 2018.
https://doi.org/10.1007/978-3-319-98355-4_3

n-long inputs with only $O(sn)$ states; so, the model is powerful enough to represent any 2DFA algorithm for OWL_h on inputs of fixed length, and remains small if so do both the 2DFA and the length. We also know that certain options for (i) and (ii) make it necessary for a RA solving OWL_h to use $2^{\Omega(h)}$ states, even when the length is restricted to just $n = 2$ symbols [3, Theorems 3 and 4]; while other options make $O(h)$ states already sufficient for $n = 2$ symbols [6], and $O(h^2)$ states already sufficient for $n = 3$ and $n = 4$ symbols [3, Theorems 6 and 7].

For example, the linear upper bound for two symbols from [6] was proved for the case where (i) there is one propositional variable $e_{a,b}$ for every two vertices a, b of the input graph, indicating whether an edge between a and b exists; and (ii) all rules of propositional-formula formation are allowed. That is, if 2OWL_h denotes the restriction of OWL_h to instances of length exactly 2, then $O(h)$ states are enough for solving 2OWL_h by a RA which builds its formulas by arbitrarily applying the logical connectives $\wedge, \vee, \neg$ to the variables $e_{a,b}$. In fact, this upper bound is also known to be optimal in the specific case, as Bianchi, Hromkovič, and Kováč [1, Theorem 1] recently proved that such RAs for 2OWL_h also need $\Omega(h)$ states. Overall, we arrive at the nice conclusion that, *with such variables and rules, every smallest* RA for 2OWL_h *has* $\Theta(h)$ *states.*

By inspecting the proof of [1, Theorem 1] for the above linear lower bound, one can observe that it is actually valid not only for RAs with the particular variables and rules, but also for all possible RAs. In fact, minor technical changes make that proof valid even for arbitrary 2DFAs. That is, every 2DFA that solves 2OWL_h needs $\Omega(h)$ states, even if it is not reasonable. At the same time, one easily realizes that the RA that proves the matching upper bound in [6] implies directly that a 2DFA, too, can solve 2OWL_h with $O(h)$ states. Overall, we actually know the much stronger fact that *every smallest* 2DFA *for* 2OWL_h *has* $\Theta(h)$ *states.*

At this point, it is interesting to ask what the corresponding fact is for OWL_h on three symbols, or four, or five, and so on. In general, for $n \geq 2$, we let $n\text{OWL}_h$ denote the restriction of OWL_h to instances of exactly n symbols, and ask:

$$\textit{How many states are there in a smallest } \text{2DFA} \textit{ for } n\text{OWL}_h\textit{?} \tag{1}$$

For $n = 2$, the *asymptotic* answer to this question is, of course, provided by our discussion above. Note, however, that we are still missing the *exact* answer, namely a function $s(h)$ such that some 2DFA for 2OWL_h has at most $s(h)$ states and no 2DFA for 2OWL_h has strictly fewer than $s(h)$ states.

For $n \geq 3$, we know neither the asymptotic nor the exact answer to (1). We only have the asympotic upper bounds implied by the RAs of [3, Theorems 6 and 7] which implement Savitch's algorithm on n symbols and are easily converted into 2DFAs with (n times more states, and thus with) the same asymptotic size (if n is constant). For the cases $n = 3$ and $n = 4$, those bounds are both quadratic, so we know that *every smallest* 2DFA *for* 3OWL_h *or for* 4OWL_h *has* $O(h^2)$ *states.*

In this paper, we study (1) for the cases $n = 2$ and $n = 3$. Our main contribution is the asymptotic answer for $n = 3$ (Sect. 4):

Theorem 1. *Every smallest* 2DFA *for* 3OWL *has* $\Theta(h^2/\log h)$ *states.*

This involves a new algorithm for the upper bound (Lemma 8) and a standard argument for the lower bound (Lemma 7). Before that, we also take the time to carefully examine the case $n = 2$ and the known asymptotic answer (Sect. 3):

Theorem 2. *Every smallest* 2DFA *for* 2OWL *has* $\Theta(h)$ *states.*

We give a detailed argument for the lower bound (Lemma 6), which mimics that of [1, Theorem 1] but applies to any 2DFA and results in a higher exact value. For completeness, we also give the known algorithm for the upper bound (Lemma 5).

In both cases, we put the extra effort to find exact values for our bounds, so that one can appreciate the gap, between lower and upper bound, where the actual size of the smallest 2DFA lies. For example, in the case $n = 2$, one sees that the $\Theta(h)$ size of the best 2DFA is actually somewhere above $\frac{1}{2}h + \frac{1}{4}\lg h - \frac{1}{2}$ and below $2h$. We also put the effort to make our intermediate lemmata as general as possible, even if this full generality is not strictly needed in our proofs. For example, the Hybrid Rule (Lemma 2) is proved for all cases, even for when the computation on the hybrid string is looping, although we only use that rule once (Lemma 6), in a case where that computation is (accepting, and thus) halting.

2 Preparation

If S is a set, then $|S|$, $\overline{S}$, and $\mathbb{P}(S)$ are respectively its size, complement, and powerset. If Σ is an alphabet, then Σ^* is the set of all strings over it and Σ^n is its subset containing only the strings of length exactly n. If $z \in \Sigma^*$ is a string, then $|z|$ and z_j are respectively its length and its j-th symbol (if $1 \leq j \leq |z|$). If $n \geq 0$, then $[n] := \{1, 2, \ldots, n\}$ is the set of the n smallest positive integers.

Problems. A (*promise*) *problem* over Σ is any pair $\mathfrak{L} = (L, \tilde{L})$ of disjoint subsets of Σ^*. Its *positive instances* are all $w \in L$, whereas its *negative instances* are all $w \in \tilde{L}$. A machine *solves* $\mathfrak{L}$ if it accepts every positive instance but no negative one. If $\tilde{L} = \overline{L}$, then we call $\mathfrak{L}$ a *language* and represent it only by L.

Let $h \geq 2$.[1] The alphabet $\Sigma_h := \mathbb{P}([h] \times [h])$ consists of all two-column directed graphs with h nodes per column and only rightward arrows (Fig. 1a). A string $w \in \Sigma_h^n$ is naturally viewed as an $(n+1)$-column graph, where every arrow connects successive columns (Fig. 1b, c); we usually index the columns from 0 to n and, for simplicity, drop the directions of the arrows. If w contains a path from the leftmost to the rightmost column (called a *live* path), then we say that w is *live*; otherwise we say that w is *dead*. The language

$$\text{OWL}_h := \{\ w \in \Sigma_h^* \ \mid\ w \text{ is live } \}$$

represents the computational task of checking that a given string in Σ_h^* contains a live path [5]. If the string is guaranteed to be of a fixed length n, then the task is best represented by the promise problem

$$n\text{OWL}_h := \big(\ \{w \in \Sigma_h^n \mid w \text{ is live}\},\ \{w \in \Sigma_h^n \mid w \text{ is dead}\}\ \big)\,.$$

[1] Here, we exclude the trivial case of height $h = 1$, so that we can divide by $\lg h \neq 0$.

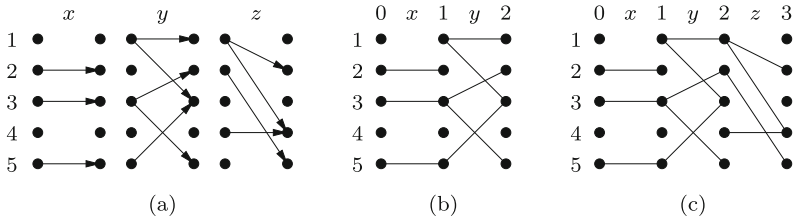

Fig. 1. (a) Three symbols $x, y, z \in \Sigma_5$; e.g., $z = \{(1,2),(1,4),(2,5),(4,4)\}$. (b) The string xy of the first two symbols, simplified and indexed, as an instance of 2OWL_5. (c) The string xyz of all symbols, simplified and indexed, as an instance of 3OWL_5.

Then $n\text{OWL} := (n\text{OWL}_h)_{h \geq 2}$ is the family of all such promise problems for $h \geq 2$.

Machines. A *two-way deterministic finite automaton* (2DFA) is any tuple of the form $M = (S, \Sigma, \delta, q_\text{s}, q_\text{a})$, where S is a set of *states*, Σ is an *alphabet*, $q_\text{s}, q_\text{a} \in S$ are respectively the *start* and *accept* states, and $\delta : S \times (\Sigma \cup \{\vdash, \dashv\}) \rightharpoonup S \times \{\text{L}, \text{R}\}$ is a (partial) *transition function*, for $\vdash, \dashv \notin \Sigma$ the left and right endmarkers, and L,R the left and right directions.

An input $w \in \Sigma^*$ is presented to M surrounded by the endmarkers, as $\vdash w \dashv$. The computation starts at q_s and on $\vdash$. In each step, the next state and head move (if any) are derived from δ and the current state and symbol. Endmarkers are never violated, except if the next state is q_a; that is, $\delta(\,.\,, \vdash)$ is always (q_a, L) or of the form $(\,.\,, \text{R})$; and $\delta(\,.\,, \dashv)$ is always (q_a, R) or of the form $(\,.\,, \text{L})$. Hence, the computation either *loops*, if it ever repeats a state on the same input cell; or *hangs*, if it ever reaches a state and symbol for which δ is undefined; or falls off $\vdash$ or $\dashv$ into q_a, in which case we say that M *accepts* w.

Formally, for any string z, position i, and state q, the *computation of M when started at q on the i-th symbol of z* is the unique sequence

$$\text{COMP}_{M,q,i}(z) = \big((q_t, i_t)\big)_{0 \leq t < m}$$

where $(q_0, i_0) = (q, i)$, $1 \leq m \leq \infty$, every pair is derived from its predecessor via δ and z, every pair is within z ($1 \leq i_t \leq |z|$) except possibly for the last one, and the last pair is within z iff δ is undefined on the corresponding state and symbol. We say m is the *length* of this computation. If $m = \infty$, then the computation *loops*. Otherwise, it *hits left into* q_{m-1}, if $i_{m-1} = 0$; or *hangs*, if $1 \leq i_{m-1} \leq |z|$; or *hits right into* q_{m-1}, if $i_{m-1} = |z|{+}1$ (Fig. 2). When $i = 1$ (respectively, $i = |z|$) we get the *left* (right) *computation of M from q* on z:

$$\text{LCOMP}_{M,q}(z) := \text{COMP}_{M,q,1}(z) \quad \text{and} \quad \text{RCOMP}_{M,q}(z) := \text{COMP}_{M,q,|z|}(z)\,.$$

The (*full*) *computation of M on z* is the typical $\text{COMP}_M(z) := \text{LCOMP}_{M,q_\text{s}}(\vdash z \dashv)$, so that M accepts z iff $\text{COMP}_M(z)$ hits right or left (into q_a).

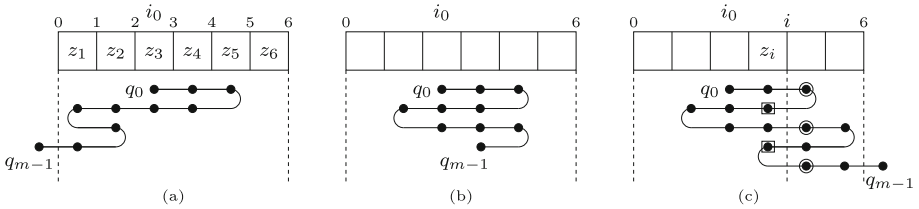

Fig. 2. (a) Cells and boundaries on a 6-long z; a computation that hits left. (b) One that hangs. (c) One that hits right, and its i-th frontier: R_i^c in circles and L_i^c in boxes.

Frontiers. Consider a computation $c = ((q_t, i_t))_{0 \le t < m}$ over some input z, and the index $0 \le i \le |z|$ of some boundary of z (Fig. 2c). How does c behave over that boundary? One answer to this question is the standard notion of the *i-th crossing sequence of c* [2], namely the sequence $q_1, q_2, q_3, \ldots$ where q_j is the state entered by c right after the j-th time that c crosses that boundary.

Another answer, with much less information, is the *i-th frontier of c* [4], which records only which states are used by the crossings, completely ignoring the order in which they are used. Formally, this is the pair of sets of states (L_i^c, R_i^c), where R_i^c (respectively, L_i^c) consists of every state which is entered by c right after some left-to-right (right-to-left) crossing of the i-th boundary of z:

$$R_i^c := \{q_t \mid 0 \le t < m \ \&\ i_{t-1} = i \ \&\ i_t = i+1\},$$
$$L_i^c := \{q_t \mid 0 \le t < m \ \&\ i_{t-1} = i \ + \ 1 \ \&\ i_t = i\}.$$

Here we also assume $i_{-1} = i_0 - 1$, so that, if c starts on the cell right after the boundary ($i_0 = i + 1$), then R_i^c also contains q_0. (This reflects the convention that the initial state q_0 is always the result of an 'invisible' left-to-right step.)

If c is a full computation, then it starts (on $\vdash$, and thus) on the left side of the i-th boundary of z (since $i \ge 0$), and then eventually hangs or loops or accepts. If it hangs or accepts also on the left side of the boundary, then it crosses it from left to right exactly as many times as it crosses it from right to left; hence, R_i^c and L_i^c contain the same number of states. By similar reasoning, if c hangs or accepts on the right side of the boundary, then R_i^c contains one more state than L_i^c. Finally, if c loops, then R_i^c contains either exactly as many states as L_i^c, if c never crosses the boundary or the latest crossing which does not result in a repeated pair (q_t, i_t) is from right to left; or one more state, otherwise. Overall, we conclude that, if c is full, then $|R_i^c|$ is always either $|L_i^c|$ or $|L_i^c|+1$.

With this motivation, we define *a frontier of M* to be any pair (L, R) such that $L, R \subseteq S$ and either $|L| = |R|$ or $|L|+1 = |R|$. In the former case, the frontier is called *balanced*; otherwise, it is called *unbalanced*. Standard counting arguments show that, if M has s states, then it has $\binom{2s}{s}$ balanced and $\binom{2s}{s+1}$ unbalanced frontiers, for a total of $\binom{2s+1}{s+1}$ frontiers overall.

The next lemma (Lemma 1) and rule (Lemma 2) are of independent interest. In this paper, we will use them to prove a lower bound for $\textsc{2owl}_h$ (Lemma 6).

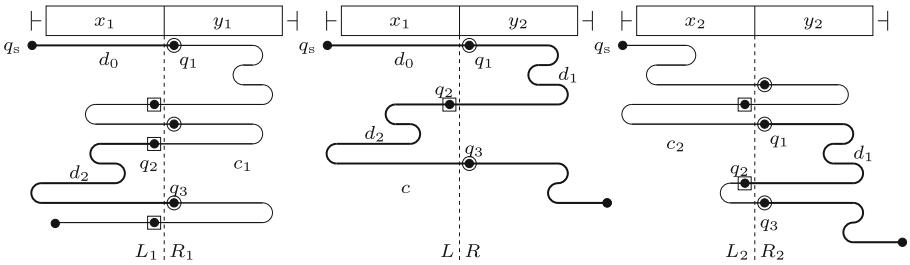

Fig. 3. Computations in the proof of the Hybrid Lemma.

Lemma 1 (Hybrid Lemma). *Let c_1, c_2, c be respectively the full computations of M on strings x_1y_1, x_2y_2 and their hybrid x_1y_2. Let $(L_1, R_1), (L_2, R_2), (L, R)$ be the respective frontiers on the boundaries x_1-y_1, x_2-y_2, and x_1-y_2. Then*

$$L_1 \supseteq L_2 \;\&\; R_1 \subseteq R_2 \quad\Longrightarrow\quad L_1 \supseteq L_2 \supseteq L \;\&\; R \subseteq R_1 \subseteq R_2\,.$$

Proof. Suppose $L_1 \supseteq L_2$ and $R_1 \subseteq R_2$. We must prove four inclusions. The first and last of them are just our assumption (included in the statement only for aesthetics), so we just need to prove that $L_2 \supseteq L \,\&\, R \subseteq R_1$.

Let $0 \leq k \leq \infty$ be the number of times c crosses the x_1-y_2 boundary. Let q_j be the state entered by c right after the j-th crossing, for all finite j with $1 \leq j \leq k$ (Fig. 3). It suffices to prove the following.

Claim. *All q_j for odd j are in R_1, and all q_j for even j are in L_2.*

Indeed, if the claim holds, then every $q \in R$ is also in R_1, because it is a q_j for some odd j; and every $q \in L$ is also in L_2, because it is a q_j for some even j.

To prove the claim, we first note that it is vacuously true when $k = 0$. So, we assume $1 \leq k \leq \infty$ and apply induction on j.

In the base case, $j = 1$ and we must prove $q_1 \in R_1$. But q_1 is the result of the first crossing of the x_1-y_2 boundary, so $d_0 := \textsc{lcomp}_{M,q_s}(\vdash x_1)$ hits right into q_1. But d_0 is clearly a prefix of c_1, therefore the first crossing of the x_1-y_1 boundary along c_1 results into q_1, as well. Hence, $q_1 \in R_1$.

In the inductive step, we assume the claim for $j \geq 1$ and prove it for $j{+}1 \leq k$.

If j is odd, then $j{+}1$ is even, and thus q_{j+1} is the result of crossing the x_1-y_2 boundary from right to left. In particular, $d_j := \textsc{lcomp}_{M,q_j}(y_2\dashv)$ is an infix of c and hits left into q_{j+1}. We know $q_j \in R_1$ (by the inductive hypothesis), and thus $q_j \in R_2$ (since $R_1 \subseteq R_2$). Therefore, c_2 produces q_j in one of its left-to-right crossings of the x_2-y_2 boundary. Hence, after that crossing, c_2 continues as in $\textsc{lcomp}_{M,q_j}(y_2\dashv)$, namely as in d_j, and thus crosses the boundary again, from right to left and into q_{j+1}. Consequently, $q_{j+1} \in L_2$.

If j is even, we work symmetrically. Since $j{+}1$ is odd and ≥ 3, q_{j+1} results from a left-to-right crossing of the x_1-y_2 boundary and c contains the infix $d_j := \textsc{rcomp}_{M,q_j}(\vdash x_1)$ which hits right into q_{j+1}. But q_j is in L_2 (by the inductive hypothesis), and thus in L_1 (since $L_1 \supseteq L_2$), so c_1 produces it in some

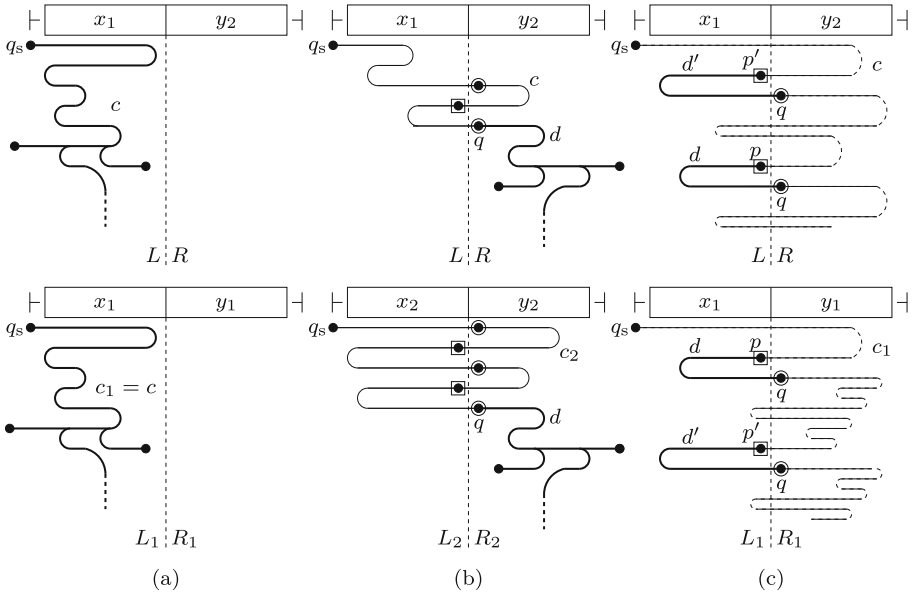

Fig. 4. Computations in the proof of the Hybrid Rule. (a) If c never crosses the critical boundary: then c, c_1 decide the same. (b) If c crosses the critical boundary finitely often, and the last crossing is from left to right: then c, c_2 decide the same. (c) If c crosses the critical boundary infinitely often, and the earliest same-side repetition is after a left-to-right crossing: then c, c_1 decide the same.

right-to-left crossing of the x_1-y_1 boundary. Hence, after that, c_1 continues as in $\textsc{rcomp}_{M,q_j}(\vdash x_1) = d_j$, which means that it left-to-right crosses the boundary into q_{j+1}, causing $q_{j+1} \in R_1$. □

Lemma 2 (Hybrid Rule). *Suppose M decides identically on strings x_1y_1, x_2y_2 but differently on their hybrid x_1y_2. Then the frontiers $(L_1, R_1), (L_2, R_2)$ of the full computations of M on x_1y_1, x_2y_2 on the boundaries x_1-y_1 and x_2-y_2 satisfy:*

$$L_1 \not\supseteq L_2 \quad \vee \quad R_1 \not\subseteq R_2\,.$$

Proof. Let c_1, c_2, c be the full computations of M on strings x_1y_1, x_2y_2, and their hybrid x_1y_2. Let $(L_1, R_1), (L_2, R_2), (L, R)$ be the frontiers of these computations on the boundaries x_1-y_1, x_2-y_2, and x_1-y_2, respectively. Towards a contradiction, assume $L_1 \supseteq L_2 \,\&\, R_1 \subseteq R_2$. Then, by the Hybrid Lemma,

$$L_1 \supseteq L_2 \supseteq L \;\;\&\;\; R \subseteq R_1 \subseteq R_2\,.$$

Using this, we will prove that on at least one of x_1y_1 and x_2y_2, the decision of M must be identical to its decision on the hybrid x_1y_2—a contradiction.

We take cases on how often c crosses the critical boundary x_1-y_2 (Fig. 4).

If c never crosses the critical boundary, then c lies fully within $\vdash x_1$ (Fig. 4a). So, M notices no difference between x_1y_2, x_1y_1, and decides identically on both.

If c crosses the critical boundary finitely often, then there is a last crossing.

Suppose this last crossing is from left to right (Fig. 4b). Let q be the state resulting from it. Then $d := \textsc{lcomp}_{M,q}(y_2\dashv)$ does not hit left (it hangs, or loops, or falls off $\dashv$), and is thus a suffix of c. At the same time, $q \in R$ (by its selection as the result of a left-to-right crossing), and thus $q \in R_2$ (since $R \subseteq R_2$). Hence, c_2 also contains a left-to-right crossing of the boundary x_2-y_2 that results in q. Clearly, from that point on, c_2 behaves as in $\textsc{lcomp}_{M,q}(y_2\dashv)$, namely as in d, which does not hit left. Therefore c_2 also finishes with d. Overall, c and c_2 have the same suffix d, which implies that M decides identically on x_1y_2 and x_2y_2.

If the last crossing of the x_1-y_2 boundary along c is from right to left, then a symmetric argument applies: $q \in L \subseteq L_1$ and $d := \textsc{rcomp}_{M,q}(\vdash x_1)$ is a suffix of both c and c_1, causing M to decide identically on x_1y_2 and x_1y_1.

If c crosses the critical boundary infinitely often, then consider the infinite list $q_1, q_2, \ldots$ where q_j is the state produced by the j-th crossing (Fig. 4c). Of course, this list contains repetitions: there exist $1 \leq j_1 < j_2$ such that $q_{j_1} = q_{j_2}$. Equally clearly, it also contains *same-side* repetitions: namely, repetitions where j_1, j_2 are either both odd or both even (so that q_{j_1}, q_{j_2} are produced both by left-to-right crossings or both by right-to-left crossings, respectively). Let q be the state in the earliest such repetition (namely $q = q_{j_1} = q_{j_2}$ in the same-side repetition with the smallest j_2) and p the state produced by the crossing just before that repetition happened (namely $p = q_{j_2-1}$).

Suppose the j_1-th and j_2-th crossings are from left to right. Then the j_2-1st crossing is from right to left and is followed by $d := \textsc{rcomp}_{M,p}(\vdash x_1)$, which hits right into q. At the same time, $p \in L$ and $q \in R$ (by their selection as the results of a right-to-left and a left-to-right crossing, respectively) and thus $p \in L_1$ and $q \in R_1$ (as $L_1 \supseteq L$ and $R \subseteq R_1$). So, c_1 also contains right-to-left crossings that produce p (to be called "p-crossings") and left-to-right crossings that produce q (to be called "q-crossings"). The next claim implies that at least one of these two types of crossings repeats, and thus c_1 loops, exactly as c does. Therefore, M decides identically on x_1y_2 and x_1y_1.

Claim. *There are at least two p-crossings or at least two q-crossings in c_1.*

Proof. Towards a contradiction, assume c_1 contains exactly one p-crossing and exactly one q-crossing. We distinguish cases based on their order inside c_1.

If the q-crossing appears before the p-crossing: We know the p-crossing is followed by $\textsc{rcomp}_{M,p}(\vdash x_1)$, namely by d, which we already know hits right into q. So, the p-crossing is followed by a q-crossing. Hence, c_1 contains at least two q-crossings (one before and one after the p-crossing), a contradiction.

If the q-crossing appears after the p-crossing: We first return to c to observe that $j_1 \neq 1$, namely the first of the two crossings that produce q cannot be the very first of all crossings. (Because then $\textsc{lcomp}_{M,q_s}(\vdash x_1)$ hits right into q; so the one q-crossing in c_1 is also the very first of all crossings, and thus appears before the p-crossing, a contradiction.)

Hence, we can talk about the (j_1-1)-st crossing of the critical boundary in c (from right to left). Let p' be the state it produces. Then $d' := \textsc{rcomp}_{M,p'}(\vdash x_1)$ hits right into q. Note that $p' \neq p$, or else $q_{j_1-1} = q_{j_2-1}$, contrary to our selection of $q = q_{j_1} = q_{j_2}$ as the earliest same-side repetition. Also note that $p' \in L$ (as the result of a right-to-left crossing), hence $p' \in L_1$ (since $L_1 \supseteq L$), hence c_1 also contains a right-to-left crossing that produces p' (to be called "p'-crossing").

It now follows that c_1 contains at least two q-crossings: the one right after the p-crossing (caused by d) and the one right after the p'-crossing (caused by d'), which we know are distinct (because $p' \neq p$). This is again a contradiction. ⊡

If the j_1-th and j_2-th crossings are from right to left, then we argue symmetrically. We know that $d := \textsc{lcomp}_{M,p}(y_2\dashv)$ hits left into q, and that $p \subseteq R \subseteq R_2$ and $L_2 \supseteq L \ni q$, so c_2 also contains left-to-right crossings that produce p and right-to-left crossings that produce q. As before, at least one of these two types of crossings repeats, hence c_2 loops, causing M to decide on x_2y_2 just as on x_1y_2.

This concludes the third case of our argument and, with it, the full proof. □

Behaviors. Let z be any string. The *behavior of M on z* is the (partial) function which returns the results of all left and right computations of M on z. Specifically, it is the function $\gamma_{M,z} : S \times \{\text{L},\text{R}\} \rightharpoonup S \times \{\text{L},\text{R}\}$ such that, for all $p \in S$,

$$\gamma_{M,z}(p,\text{L}) := \begin{cases} (q,\text{L}) & \text{if } \textsc{lcomp}_{M,p}(z) \text{ hits left into } q, \\ \text{undefined} & \text{if } \textsc{lcomp}_{M,p}(z) \text{ hangs or loops}, \\ (q,\text{R}) & \text{if } \textsc{lcomp}_{M,p}(z) \text{ hits right into } q; \end{cases}$$

and similarly for $\gamma_{M,z}(p,\text{R})$, using $\textsc{rcomp}_{M,p}(z)$ instead of $\textsc{lcomp}_{M,p}(z)$.

With this motivation, we define a *behavior of M* to be any partial function from $S\times\{\text{L},\text{R}\}$ to $S\times\{\text{L},\text{R}\}$. Easily, if M has s states, then it has $(2s+1)^{2s}$ behaviors. However, if $|z| = 1$, then every left computation is also a right computation (since the first and last cells of z coincide), causing $\gamma_{M,z}(p,\text{L}) = \gamma_{M,z}(p,\text{R})$ for all p; hence, the number of *single-symbol behaviors* of M is only $(2s+1)^s$.

A standard fact in the analysis of 2DFA computations is that, if M exhibits the same behavior on two strings, then each of them can be replaced by the other in any context, without M noticing the difference. Formally, this is captured by the next lemma, which we state without proof.

Lemma 3 (Infix Lemma). *If $\gamma_{M,y_1} = \gamma_{M,y_2}$, then $\gamma_{M,xy_1z} = \gamma_{M,xy_2z}$.*

As a direct consequence of this, M cannot decide differently on two strings which differ only at two infixes that do not change its behavior.

Lemma 4 (Infix Rule). *If M decides differently on strings xy_1z, xy_2z, then:*

$$\gamma_{M,y_1} \neq \gamma_{M,y_2}\,.$$

Proof. Towards the contrapositive, assume $\gamma_{M,y_1} = \gamma_{M,y_2}$. Then, by the Infix Lemma, $\gamma_{M,\vdash xy_1z\dashv} = \gamma_{M,\vdash xy_2z\dashv}$. In particular, the two functions return the same value on (q_s,L), which is either (q_a,L), or (q_a,R), or undefined. In all three cases, it follows that M decides identically on xy_1z and xy_2z. □

3 The Case of Two Symbols

We now prove Theorem 2, that *all smallest* 2DFAs for 2OWL_h have $\Theta(h)$ *states*. This follows directly from the next two lemmata. The upper bound (Lemma 5) is well-known and also implied by [6]; here, we give a careful construction. The lower bound (Lemma 6) is a tighter variant of [1, Theorem 1] that uses frontiers, as opposed to arbitrary pairs of sets of states. (Without this modification, the lower bound for 2DFAs by the argument of [1, Theorem 1] is only $\frac{1}{2}h$.)

Lemma 5. *Some* 2DFA *solves* 2OWL_h *with* $\leq 2h$ *states.*

Proof. Fix $h \geq 2$ and consider an instance xy of 2OWL_h, for $x, y \in \Sigma_h$ (Fig. 1b). Let $u_1, u_2, \ldots, u_h$ be the nodes of column 1, from top to bottom. We say u_i is L-*live*, if it has non-zero degree in x; R-*live*, if it has non-zero degree in y; *live*, if it is both L-live and R-live; and *dead* if it is not live. Clearly, xy is live iff some u_i is live. So, our 2DFA M simply searches for a live u_i sequentially, from u_1 to u_h.

The set of states is $S := [h] \times \{\text{L},\text{R}\}$ and state $(1, \text{L})$ serves as both the start and the accept state. Each other state (i, L) is used only on $\vdash x$; it assumes that all u_j above u_i are dead and that u_i is R-live, and tries to check if u_i is also L-live. Symmetrically, every state (i, R) is used only on $y\dashv$; it assumes that all u_j above u_i are dead and that u_i is L-live, and tries to check if u_i is also R-live.

The transitions between these states are now not hard to see:

From $(1, \text{L})$ on $\vdash$, M moves to $(1, \text{L})$ on x. If x contains no edges, then xy is obviously dead, so M just hangs. Otherwise, there exists at least one L-live u_i, so M finds the topmost such u_i and moves to the corresponding state (i, R) on y.

From a state (i, R) on y, M checks if u_i is R-live. If so, then xy is live, so M moves to (i, R) on $\dashv$, and then off $\dashv$ into $(1, \text{L})$ to accept. Otherwise, it checks if any u_j below u_i is R-live. If not, then xy is dead, so M just hangs. Otherwise, M finds the topmost R-live u_j below u_i, and moves to the corresponding state (j, L) on x, to check whether that u_j is also L-live.

From a state (i, L) on x with $i \geq 2$, M behaves symmetrically as above: if u_i is L-live, then M moves to (i, L) on $\vdash$, and then off $\vdash$ into $(1, \text{L})$ to accept. Otherwise, it either hangs, if no u_j below u_i is L-live; or moves to y and into the state (j, R) corresponding to the topmost such u_j.

It should be clear that M works correctly and uses exactly $2h$ states. □

Lemma 6. *Every* 2DFA *solving* 2OWL_h *has* $> \frac{1}{2}h + \frac{1}{4}\lg h - \frac{1}{2}$ *states.*

Proof. Let M be any 2DFA solving 2OWL_h. Let S be its set of states and $s := |S|$.

For every $\alpha \subseteq [h]$, let $x_\alpha := \{(u, u) \mid u \in \alpha\}$ be the symbol consisting of the "horizontal" edges which correspond to indices in α (e.g., see the leftmost symbol in Fig. 1a). Clearly, for every $\alpha, \beta \subseteq [h]$, the string $x_\alpha x_\beta$ is live iff $\alpha \cap \beta \neq \emptyset$. We also easily verify the following.

Claim 1. *For all distinct* $\alpha, \beta \subseteq [h]$:

(i) the strings $x_\alpha x_{\overline{\alpha}}$ *and* $x_\beta x_{\overline{\beta}}$ *are dead, but*
(ii) at least one of their two hybrids $x_\alpha x_{\overline{\beta}}$ *and* $x_\beta x_{\overline{\alpha}}$ *is live.*

Proof. Let $\alpha, \beta \subseteq [h]$ with $\alpha \neq \beta$. Then (i) is obvious, since $\alpha \cap \overline{\alpha} = \beta \cap \overline{\beta} = \emptyset$. For (ii), suppose both hybrids $x_\alpha x_{\overline{\beta}}$ and $x_\beta x_{\overline{\alpha}}$ are dead. Then $\alpha \cap \overline{\beta} = \emptyset$ and $\beta \cap \overline{\alpha} = \emptyset$; equivalently, $\alpha \subseteq \beta$ and $\beta \subseteq \alpha$; hence $\alpha = \beta$, a contradiction. ⊡

Now, for each $\alpha \subseteq [h]$, consider the the full computation of M on $x_\alpha x_{\overline{\alpha}}$ and let (L_α, R_α) be its frontier over the middle boundary. This effectively defines 2^h frontiers of M, one for each α. We claim that these are all distinct.

Claim 2. *For all distinct* $\alpha, \beta \subseteq [h]$: $(L_\alpha, R_\alpha) \neq (L_\beta, R_\beta)$.

Proof. Let $\alpha, \beta \subseteq [h]$ with $\alpha \neq \beta$. By Claim 1 and since M solves $\textsc{2owl}_h$, we know M decides identically on $x_\alpha x_{\overline{\alpha}}$ and $x_\beta x_{\overline{\beta}}$ (it does not accept) but differently on at least one of their hybrids $x_\alpha x_{\overline{\beta}}$ and $x_\beta x_{\overline{\alpha}}$ (it accepts). Without loss of generality, assume the interesting hybrid is $x_\alpha x_{\overline{\beta}}$ (otherwise, swap the roles of the two strings). Then, by the Hybrid Rule (Lemma 2), we know $L_\alpha \not\supseteq L_\beta$ or $R_\alpha \not\subseteq R_\beta$. Therefore $L_\alpha \neq L_\beta$ or $R_\alpha \neq R_\beta$, namely $(L_\alpha, R_\alpha) \neq (L_\beta, R_\beta)$. ⊡

Overall, we conclude that M uses distinct frontiers on the 2^h distinct dead strings of the form $x_\alpha x_{\overline{\alpha}}$. Since it has only $\binom{2s+1}{s+1}$ frontiers total, it follows that

$$\binom{2s+1}{s+1} \geq 2^h \,, \tag{2}$$

and we need to solve for s. Using the well-known Stirling's bounds

$$\sqrt{2\pi}\sqrt{n} \cdot (\tfrac{n}{e})^n \;\leq\; n! \;\leq\; e\sqrt{n} \cdot (\tfrac{n}{e})^n$$

for the factorial function, we calculate:

$$\binom{2s+1}{s+1} = \frac{(2s+1)!}{(s+1)!s!} = \frac{2s+1}{s+1} \cdot \frac{(2s)!}{(s!)^2} < 2 \cdot \frac{e\sqrt{2s} \cdot (2s/e)^{2s}}{(\sqrt{2\pi}\sqrt{s} \cdot (s/e)^s)^2} = \frac{e\sqrt{2}}{\pi} \cdot \frac{2^{2s}}{\sqrt{s}}$$

so that (2) implies $(e\sqrt{2}/\pi)(2^{2s}/\sqrt{s}) > 2^h$, and thus

$$s > \tfrac{1}{2}h + \tfrac{1}{4}\lg s - \tfrac{1}{2}\lg(e\sqrt{2}/\pi) \,. \tag{3}$$

Since $s \geq 1$ and $\frac{1}{2}\lg(e\sqrt{2}/\pi) \leq 0.15$, this implies $s > \frac{1}{2}h - 0.15$, and thus $s \geq \frac{1}{2}h$ (since s and h are both integers). So, $\lg s \geq \lg h - 1$. Substituting in (3), we get:

$$s > \tfrac{1}{2}h + \tfrac{1}{4}\lg h - \tfrac{1}{2}\lg(2e/\pi) \,.$$

Finally, we note that $\lg(2e/\pi) \approx 0.8 < 1$. □

4 The Case of Three Symbols

To prove Theorem 1, that *all smallest* 2DFAs for $\textsc{3owl}_h$ *have* $\Theta(h^2/\log h)$ *states*, we start with the lower bound (Lemma 7), which is simpler; then continue with the upper bound (Lemma 8), which is a bit more involved.

Lemma 7. *Every* 2DFA *solving* 3OWL$_h$ *has* $\geq \frac{1}{2}\frac{h^2}{\lg h}$ *states.*

Proof. Let M be any 2DFA solving 3OWL$_h$. Let S be its set of states and $s := |S|$.

Let $y_1, y_2, \ldots, y_N$ be a list of all symbols in the input alphabet Σ_h. Since every symbol may contain up to h^2 edges, we know $N = 2^{h^2}$. For each $i = 1, 2, \ldots, N$, let $\gamma_i := \gamma_{M,y_i}$ be the behavior of M on y_i.

Claim. *For all distinct* $i, j \in [N]$: $\gamma_i \neq \gamma_j$.

Proof. Let $i, j \in [N]$ with $i \neq j$. Then $y_i \neq y_j$. Hence, there exists at least one edge (u, v) which appears in one of y_i, y_j, but not the other. Without loss of generality, assume (u, v) appears in y_i but not in y_j. Let $x := \{(u, u)\}$ and $z := \{(v, v)\}$ be the symbols containing only the "horizontal" edges corresponding to u and v. Then the strings xy_iz and xy_jz are respectively a live and a dead instance of 3OWL$_h$. Hence, M decides differently on them. By the Infix Rule (Lemma 4), it follows that $\gamma_i \neq \gamma_j$. ⊡

Hence, M uses distinct behaviors on the 2^{h^2} possible middle symbols. Since it has only $(2s+1)^s$ single-symbol behaviors in total (cf. p. 9), it follows that

$$(2s+1)^s \geq 2^{h^2}\,, \tag{4}$$

which implies the bound in the statement, as follows.

If $h = 2$, then (4) asks that $(2s+1)^s \geq 16$. Easily, this holds only if $s \geq 2$, which matches the bound in the statement: $\frac{1}{2}(h^2/\lg h) = \frac{1}{2}(4/1) = 2$.

If $h = 3$, then similarly we need $(2s+1)^s \geq 512$, which holds only if $s \geq 4$, which exceeds the bound in the statement: $\frac{1}{2}(h^2/\lg h) = \frac{1}{2}(9/\lg 3) \approx 2.84$.

If $h \geq 4$, then we first take logarithms to rewrite (4) as $s\lg(2s+1) \geq h^2$. Towards a contradiction, we assume $s < \frac{1}{2}\frac{h^2}{\lg h}$ and calculate:

$$\lg(2s+1) < \lg\big(2 \cdot 2s\big) < \lg\big(2\tfrac{h^2}{\lg h}\big) = 2\lg h - (\lg\lg h - 1) \leq 2\lg h\,,$$

where the first step uses the fact that $2s+1 < 4s$ (since $s \geq 1$); the second step uses the assumption that $s < \frac{1}{2}\frac{h^2}{\lg h}$; and the last step uses the fact that $\lg\lg h \geq 1$ (since $h \geq 4$). Therefore, we can conclude that

$$s\lg(2s+1) < \big(\tfrac{1}{2}\tfrac{h^2}{\lg h}\big)\big(2\lg h\big) = h^2\,,$$

contrary to the rewriting of (4) above. □

Lemma 8. *Some* 2DFA *solves* 3OWL$_h$ *with* $\leq 4h\lceil\frac{h}{\lfloor\lg h\rfloor}\rceil$ *states.*

Proof. Fix $h \geq 2$ and consider an instance xyz of 3OWL$_h$ for $x, y, z \in \Sigma_h$ (Fig. 1c). Let $u_1, u_2, \ldots, u_h$ and $v_1, v_2, \ldots, v_h$ be the nodes of columns 1 and 2, respectively, from top to bottom. Similarly to the proof of Lemma 5, we say u_i is L-*live* if it has non-zero degree in x; and v_i is R-*live* if it has non-zero degree in z.

We first partition column 1 into $h/\lg h$ blocks of length $\lg h$ each. More carefully, we let $l := \lfloor\lg h\rfloor$ be the desired length, and assign every u_i to block $b_{\lceil i/l\rceil}$.

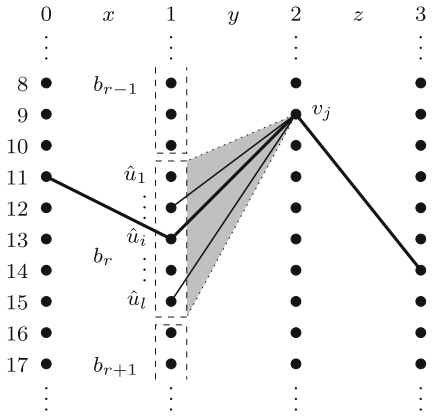

Fig. 5. The proof of Lemma 8 for the example $h = 32$, $l = 5$, $m = 7$, and $r = 3$. Relative to b_r, the shown vertex v_j has connectivity $\{2, 3, 5\}$.

Easily, this produces $m := \lceil h/l \rceil$ disjoint blocks, $b_1, b_2, \ldots, b_m \subseteq \{u_1, u_2, \ldots, u_h\}$. For example, for $h = 5$, the desired length is $l = 2$ and we get the $m = 3$ blocks $b_1 = \{u_1, u_2\}$, $b_2 = \{u_3, u_4\}$, and $b_3 = \{u_5\}$. Note that, if l does not divide h, then the length of the last block b_m is not l, but only $l' := h \bmod l$.

We say a block b_r is *live* if there exists a live path that passes through a node belonging to b_r (Fig. 5). Clearly, xyz is live iff some b_r is live. So, our 2DFA M simply searches for a live b_r sequentially, from b_1 to b_m.

Specifically, M consists of m disjoint sub-automata $M_1, M_2, \ldots, M_m$, where every M_r uses states of the form $(r, \ldots)$ and is responsible for checking whether the corresponding b_r is live. The start state of M is the start state of M_1. From then on, the iteration works as follows. If the current M_r confirms that b_r is live, then it falls off $\dashv$ into the designated accept state q_a of M, causing M to accept, too. Otherwise, M_r arrives at a state and cell where it has confirmed that b_r is dead; from there, it either advances the iteration by moving to x and to the start state of M_{r+1}, if $r < m$; or just hangs, causing M to hang too, if $r = m$.

Every M_r works in the same way as every other, except that it focuses on its own b_r. So, all sub-automata are of the same size $\tilde{s}$, causing M's total size to be $s = m\tilde{s} = O(h\tilde{s}/\log h)$. Hence, to achieve the $O(h^2/\log h)$ size in the statement, we ensure that each M_r has size $O(h)$. Here is how.

Checking a block. Fix any b_r and let $\hat{u}_1, \hat{u}_2, \ldots, \hat{u}_l$ be its vertices, from top to bottom. In the case where b_r is the last block b_m and has only $l' < l$ nodes, we pretend that the nodes $\hat{u}_{l'+1}, \ldots, \hat{u}_l$ exist but have degree zero in both x and y.

Now consider any v_j in column 2. (See Fig. 5.) The *connectivity of* v_j (*relative to* b_r) is the set $\xi \subseteq [l]$ of the indices of all $\hat{u}_i$ which connect to v_j:

$$i \in \xi \iff y \text{ contains the edge } (\hat{u}_i, v_j)\,.$$

With this motivation, we call *connectivity* any subset of $[l]$. Clearly, the number of different connectivities is $k := 2^l = 2^{\lfloor \lg h \rfloor} \leq 2^{\lg h} = h$. Let $\xi_1, \xi_2, \ldots, \xi_k$ be any fixed listing of them, so that each v_j has its connectivity in this list.

We say that, relative to b_r, a connectivity ξ_t is L-*live*, if for at least one $i \in \xi_t$ the corresponding $\hat{u}_i$ is L-*live*; R-*live*, if at least one of the v_j with connectivity ξ_t is R-*live*; *live*, if it is both L-live and R-live; and *dead*, if it is not live. Easily:

Claim. *Block b_r is live iff at least one connectivity ξ_t is live relative to it.*

Proof. Suppose b_r is live. Then some live path passes through it. Let $\hat{u}_i$ and v_j be the nodes used by this path in b_r and in column 2, respectively, and ξ_t the connectivity of v_j relative to b_r. Then $\hat{u}_i$ is L-live and v_j is R-live (clearly); and $i \in \xi_t$ (because of the edge from $\hat{u}_i$ to v_j). Hence ξ_t is both L-live (because of $\hat{u}_i$) and R-live (because of v_j). So, ξ_t is live.

Conversely, suppose some ξ_t is live relative to b_r. Let $\hat{u}_i$ and v_j be witnesses for why it is L-live and R-live, respectively. Then (i) $\hat{u}_i$ is L-live and (ii) $i \in \xi_t$; and (iii) v_j is R-live and (iv) v_j has connectivity ξ_t relative to b_r. By (ii) and (iv), we know the edge $(\hat{u}_i, v_j)$ exists. By (i) and (iii), we know this edge is actually on a live path through $\hat{u}_i$ and v_j. Hence, b_r is live. ⊡

Therefore, our sub-automaton M_r simply searches for a live ξ_t sequentially, from ξ_1 to ξ_k. Intuitively, M_r consists of k sub-automata $M_{r,1}, M_{r,2}, \ldots, M_{r,k}$, where every $M_{r,t}$ is responsible for checking whether the corresponding ξ_t is live. The machine starts on x and in the start state of $M_{r,1}$. From then on, the iteration works as follows. If the current $M_{r,t}$ confirms that ξ_t is live, then it falls off $\dashv$ into q_{a}, causing M_r to behave exactly as promised above. Otherwise, $M_{r,t}$ "*halts*" on x or y, in the sense that it enters a state where it has confirmed that ξ_t is dead. From there: if $t < k$, then it advances the iteration over all connectivities by moving to x and to the start state of $M_{r,t+1}$; otherwise ($t = k$), it either moves to x and to the start state of M_{r+1}, if $r < m$, or just hangs, if $r = m$, causing M_r to behave exactly as promised above.

Checking a connectivity. Every $M_{r,t}$ starts on x in state (r, t, L), trying to decide whether ξ_t is L-live. For this, it checks if any of the $\hat{u}_i$ in b_r with $i \in \xi_t$ have non-zero degree in x. If not, then ξ_t is dead, so $M_{r,t}$ *halts* (on x). Else, it moves to y in state (r, t, R), to check if ξ_t is also R-live. Reading y, it computes the connectivities of all v_j relative to b_r, and focuses only on those v_j with connectivity ξ_t. If no such v_j exist, then ξ_t is dead, so $M_{r,t}$ again *halts* (on y). Else, $M_{r,t}$ iterates over all such v_j, from top to bottom, moving to z to check if any of them is R-live.

Let us first see a naive way to implement this last iteration: From y and in state (r, t, R), $M_{r,t}$ finds the topmost v_j with connectivity ξ_t and moves to z in state $(r, t, j, \top)$. Reading z, it checks whether v_j is R-live. If so, then it moves to $\dashv$ into q_{a} and then off $\dashv$ into q_{a}. Otherwise, it moves back to y in a state $(r, t, j, \bot)$. Reading y, it finds the topmost $v_{j'}$ with $j' > j$ and connectivity ξ_t. If none exists, then it *halts* (on y). Else, it moves to z in state $(r, t, j', \top)$, and so on. Of course, this implementation needs $2h$ states in each $M_{r,t}$ (two for each $j = 1, \ldots, h$), causing the total size of M_r to rise to $\Theta(h^2)$—when our goal is only $O(h)$.

The smart implementation works similarly to the naive one, except for the following modification. Every time $M_{r,t}$ moves to z to examine the R-liveness of some v_j, its state is not $(r,t,j,\top)$, but just $(r,j,\top)$. In other words, whenever on z, the automaton "forgets" the connectivity ξ_t that it is responsible for. Of course, this creates no problem in checking whether v_j is R-live (since this needs only j). Still, if v_j is not R-live and the automaton returns to y, then it is unclear how it will continue its iteration over the remaining nodes of connectivity ξ_t (below v_j), let alone the iteration over the remaining connectivities (after ξ_t). How will it manage to do all this, if it has forgotten ξ_t?

The answer is that the automaton can recover the forgotten ξ_t from (i) what it still remembers and (ii) the edges in y. Specifically, whenever v_j is not R-live, the automaton moves from z and state $(r,j,\top)$ back on y and in state $(r,j,\bot)$. Reading y, it finds all edges between the nodes of b_r and v_j; from these, it computes the connectivity of v_j relative to b_r, which is exactly ξ_t. Hence, having remembered ξ_t, it continues its iteration as if it had never forgotten it.

Put another way, the trick is that $M_{r,t}$ is not disjoint from the rest of the sub-automata $M_{r,\cdot}$. Although it uses its own states (r,t,L) and (r,t,R) on xy, it shares with the rest of the $M_{r,\cdot}$ the states $(r,j,\bot)$ and $(r,j,\top)$ on yz.

Overview. Returning to M, we see that it has states of the form (r,t,L), (r,t,R), $(r,j,\bot)$, and $(r,j,\top)$, where $r \in [m]$, $t \in [k]$, $j \in [h]$. Namely, its set of states is

$$S := ([m] \times [k] \times \{\mathrm{L},\mathrm{R}\}) \cup ([m] \times [h] \times \{\bot,\top\}),$$

for a total size of $2mk + 2mh \leq 4mh$, as promised in the statement.

State $(1,1,\mathrm{L})$ is simultaneously the start and accept state: $q_\mathrm{s} = q_\mathrm{a} := (1,1,\mathrm{L})$. When in it and on $\vdash$ or $\dashv$, the machine stays in the state and moves right (either to x, to start the search for a live block; or off $\dashv$, to accept). This is the only state used on $\vdash$ and $\dashv$. All other states are used only on x, if of the form (r,t,L); or only on y, if of the form (r,t,R) or $(r,j,\bot)$; or only on z, if of the form $(r,j,\top)$.

When in (r,t,L) reading x, the machine checks if some node in b_r with non-zero degree in x has its index in ξ_t (i.e., ξ_t is L-live relative to b_r). If so, then M moves to y and in (r,t,R) to check if ξ_t is also R-live relative to b_r. Otherwise, M "*continues the search from* x", meaning that: it moves to (stays on) x and enters $(r,t{+}1,\mathrm{L})$ to advance the iteration over connectivities, if $t < k$; or moves to (stays on) x and enters $(r{+}1,1,\mathrm{L})$ to advance the iteration over blocks, if $t = k$ and $r < m$; or hangs immediately to signify rejection, if $t = k$ and $r = m$.

When in (r,t,R) reading y, the machine checks if any node of column 2 has connectivity ξ_t relative to b_r. If so, then M finds the topmost such node v_j and moves to z and in $(r,j,\top)$ to check if v_j is R-live. Otherwise (ξ_t is not R-live), M *continues the search from* x (as above).

When in $(r,j,\top)$ reading z, the machine checks if node v_j has non-zero degree in z. If so (b_r is live), then M moves to $\dashv$ and in q_a to signify acceptance. Else, it moves back to y in $(r,j,\bot)$ to advance the iteration over nodes which share with v_j the same connectivity relative to b_r.

When in $(r, j, \bot)$ reading y, the machine finds the connectivity ξ_t of node v_j relative to b_r and searches for the topmost $v_{j'}$ with $j' > j$ and the same connectivity relative to b_r. If such $v_{j'}$ exists, then M moves to z and in $(r, j', \top)$ to check if $v_{j'}$ is R-live. Otherwise, M *continues the search from* x (as above).

This concludes the definition of the transition function of M. Note that our description uses transitions (out of the states (r, t, L)) which do not move the input head, contrary to our definition of 2DFAs as machines which always move (Sect. 2); but this violation can be easily removed, with a standard, well-known technique that does not change the number of states.

Other than that, it should be clear that M decides 3OWL_h, as promised. □

5 Conclusion

Motivated by recent work on *reasonable automata* [1,3], we initiated a study of the size of arbitrary 2DFAs solving *one-way liveness* on inputs of constant length. We gave (i) a more detailed proof of the known fact (from [1]) that $\Theta(h)$ states are necessary and sufficient for length 2; and (ii) a proof of the new fact that $\Theta(h^2/\log h)$ states are necessary and sufficient for length 3. This concludes the discussion for these two lengths and for the asymptotic size.

For longer lengths, the question remains open. For example, we can still ask: *What is the size of a smallest* 2DFA *solving one-way liveness on four symbols*? The answer is known to be both $O(h^2)$ and $\Omega(h^2/\log h)$ (by the application of Savitch's method in [3, Theorems 6 and 7]; and our Lemma 7, which clearly also extends to four symbols). However, the tight asymptotic growth remains elusive.

At the same time, we still do not know the exact sizes of the smallest 2DFAs for two and three symbols. Finding those sizes would also be very interesting.

References

1. Bianchi, M.P., Hromkovič, J., Kováč, I.: On the size of two-way reasonable automata for the liveness problem. In: International Conference on Developments in Language Theory, pp. 120–131 (2015)
2. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Boston (1979)
3. Hromkovič, J., Královič, R., Královič, R., Štefanec, R.: Determinism vs. nondeterminism for two-way automata: representing the meaning of states by logical formulæ. Int. J. Found. Comput. Sci. **24**(7), 955–978 (2013)
4. Kapoutsis, C.: Removing bidirectionality from nondeterministic finite automata. In: Jędrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 544–555. Springer, Heidelberg (2005). https://doi.org/10.1007/11549345_47
5. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Symposium on the Theory of Computing, pp. 275–286 (1978)
6. Štefanec, R.: Semantically restricted two-way automata. Ph.D. thesis, Comenius University, Bratislava (2014)