From Bidirectionality to Alternation

Nir Piterman[1] and Moshe Y. Vardi[2*]

[1] Weizmann Institute of Science, Department of Computer Science, Rehovot 76100, Israel
Email: nirp@wisdom.weizmann.ac.il, URL: http://www.wisdom.weizmann.ac.il/~nirp
[2] Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.
Email: vardi@cs.rice.edu, URL: http://www.cs.rice.edu/~vardi

Abstract. We describe an explicit simulation of 2-way nondeterministic automata by 1-way alternating automata with quadratic blow-up. We first describe the construction for automata on finite words, and extend it to automata on infinite words.

1 Introduction

The theory of finite automata is one of the fundamental building blocks of theoretical computer science. As the basic theory of finite-state systems, this theory is covered in numerous textbooks and in any basic undergraduate curriculum in computer science. Since its introduction in the 1950's, the theory had numerous applications in practically all branches of computer science, from the construction of electrical circuits [Koh70], to the design of lexical analyzers [JPAR68], and to the automated verification of hardware and software designs [VW86].

From its very inception, one fundamental theme in automata theory is the quest for understanding the relative power of the various constructs of the theory. Perhaps the most fundamental result of automata theory is the robustness of the class of regular languages, the class of languages definable by means of finite automata. Rabin and Scott showed in their classical paper that neither nondeterminism nor bidirectionality changes the expressive power of finite automata; that is, nondeterministic 2-way automata and deterministic 1-way automata have the same expressive power [RS59]. This robustness was later extended to alternating automata, which can switch back and forth between existential and universal modes (nondeterminism is an existential mode) [BL80,CKS81,LLS84].

In view of this robustness, the concept of relative expressive power was extended to cover also succinctness of description. For example, it is known that nondeterministic automata and two-way automata are exponentially more succinct than deterministic automata. The language $L_n = \{uv : u, v \in \{0,1\}^n \text{ and } u \neq v\}$ can be expressed using a 1-way nondeterministic automaton or a 2-way deterministic automaton of size polynomial in n, but a 1-way deterministic automaton accepting L_n must be of exponential size (cf. [SS78]). Alternating automata, in turn, are doubly exponentially more succinct than deterministic automata [BL80,CKS81].

* Supported in part by NSF grants CCR-9700061 and CCR-9988322, and by a grant from the Intel Corporation.

Consequently, a major line of research in automata theory is establishing tight simulation results between different types of automata. For example, given a 2-way automaton with n states, Shepherdson showed how to construct an equivalent 1-way automaton with $2^{O(n\ log(n))}$ states [She59]. Birget showed how to construct an equivalent 1-way automaton with 2^{3n} states [Bir93] (see also [GH96]). Vardi constructed the *complementary* automaton, an automaton accepting the words rejected by the 2-way automaton, with 2^{2n} states [Var89]. Birget also showed, via a chain of reductions, that a 2-way nondeterministic automaton can be converted to a 1-way alternating automaton with quadratic blow-up [Bir93]. As the converse effi cient simulation is impossible [LLS84], alternation is more powerfull than bidirectionality.

Our focus in this paper is on simulation of bidirectionality by alternation. The interest in bidirectionality and alternation in not merely theoretical. Both constructs have been shown to be useful in automated reasoning. For example, reasoning about modal μ-calculus with past temporal connectives requires alternation and bidirectionality [Str82,Var88,Var98]. Recently, model checking of specifi cations in μ-calculus on context-free and prefi x-recognizable systems has been reduced to questions about 2-way automata [KV00]. In a different fi eld of research, 2-way automata were used in query processing over semistructured data [CdGLV00].

We found Birget's construction, simulating bidirectionality by alternation with quadratic blow-up, unsatisfactory. As noted, his construction is indirect, using a chain of reductions. In particular, it uses the reverse language and, consequently, can not be extended to automata on infi nite words. The theory of fi nite automata on infi nite objects was established in the 1960s by Büchi, McNaughton and Rabin [Büc62,McN66,Rab69]. They were motivated by decision problems in mathematical logic. More recently, automata on infi nite words have shown to be useful in computer-aided verifi cation [Kur94,VW86]. We note that bidirectionality does not add expressive power also in the context of automata on infi nite words. Vardi has already shown that given a 2-way nondeterministic Büchi automaton with n states one can construct an equivalent 1-way nondeterministic Büchi with $2^{O(n^2)}$ states [Var88].

Our main result in this paper is a direct quadratic simulation of bidirectionality by alternation. Given a 2-way nondeterministic automaton with n states, we construct an equivalent 1-way alternating automaton with $O(n^2)$ states. Unlike Birget's construction, our construction is explicit. This has two advantages. First, one can see exactly how alternation can effi ciently simulate bidirectionality. (In order to convert the nondeterministic automaton into an alternating automaton we use the fact that the run of the 2-way nondeterministic automaton looks like a tree of "zigzags". We analyze the form such a tree can take and recognize, using an alternating automaton, when such a tree exists.) Second, the explicitness of the construction enables us to extend it to Büchi automata. (In the full version we also give a construction for 2-way nondeterministic Rabin and parity automata.) Since it is known how to simulate alternating Büchi automata by nondeterministic Büchi automata with exponential blow-up [MH84], our construction provides another proof of the result that a 2-way nondeterministic Büchi automaton with n states can be simulated by a 1-way nondeterministic Büchi with $2^{O(n^2)}$ states [Var88].

2 Preliminaries

We consider finite or infinite sequences of symbols from some finite alphabet Σ. Given a *word* w, an element in $\Sigma^* \cup \Sigma^\omega$, we denote by w_i the i^{th} letter of the word w. The *length* of w is denoted by $|w|$ and is defined ω for infinite words.

A *2-way nondeterministic automaton* is $A = \langle \Sigma, S, S_0, \rho, F \rangle$, where Σ is the finite alphabet, S is the finite set of states, $S_0 \subseteq S$ is the set of initial states, $\rho : S \times \Sigma \to 2^{S \times \{-1,0,1\}}$ is the transition function, and F is the acceptance set. We can run A either on finite words (*2-way nondeterministic finite automaton* or *2NFA* for short) or on infinite words (*2-way nondeterministic Büchi automaton* or *2NBW* for short). In Appendix A we show that we can restrict our attention to automata whose transition function is of the form $\rho : S \times \Sigma \to 2^{S \times \{-1,1\}}$.

A *run* on a finite word $w = w_0, ..., w_l$ is a finite sequence of states and locations $(q_0, i_0), (q_1, i_1), ..., (q_m, i_m) \in (S \times \{0, ..., l+1\})^*$. The pair (q_j, i_j) represents the automaton is in state q_j reading letter i_j. Formally, $q_0 = s_0$ and $i_0 = 0$, and for all $0 \leq j < m$, we have $i_j \in \{0, ..., l\}$ and $i_m \in \{0, ..., l+1\}$. Finally, for all $0 \leq j < m$, we have $(q_{j+1}, i_{j+1} - i_j) \in \delta(q_j, w_{i_j})$. A run is *accepting* if $i_m = l + 1$ and $q_m \in F$.

A *run* on an infinite word $w = w_0, w_1, ...$ is defined similarly as an infinite sequence. The restriction on the locations is removed (for all j, the location i_j can be every number in $\mathbb{N}$). In 2NBW, a run is *accepting* if it visits $F \times \mathbb{N}$ infinitely often. A word w is *accepted* by A if it has an accepting run over w. The *language* of A is the set of words accepted by A, denoted by $L(A)$.

In the finite case we are only interested in runs in which the same state in the same position do no repeat twice during the run. In the infinite case we minimize the amount of repetition to the unavoidable minimum. A run $r = (s_0, 0), (s_1, i_1), (s_2, i_2), ..., (s_m, i_m)$ on a finite word is *simple* if for all j and k such that $j < k$, either $s_j \neq s_k$ or $i_j \neq i_k$. A run $r = (s_0, 0), (s_1, i_1), (s_2, i_2), ...$ on an infinite word is *simple* if one of the following holds (1) For all $j < k$, either $s_j \neq s_k$ or $i_j \neq i_k$. (2) There exists $l, m \in \mathbb{N}$ such that for all $j < k < l + m$, either $s_j \neq s_k$ or $i_j \neq i_k$, and for all $j \geq l$, $s_j = s_{j+m}$ and $i_j = i_{j+m}$. In Appendix B we show that there exists an accepting run iff there exists a simple accepting run. Hence, it is enough to consider simple accepting runs.

Given a set S we first define the set $B^+(S)$ as the set of all positive formulas over the set S with 'true' and 'false' (i.e., for all $s \in S$, s is a formula and if f_1 and f_2 are formulas, so are $f_1 \wedge f_2$ and $f_1 \vee f_2$). We say that a subset $S' \subseteq S$ *satisfies* a formula $\varphi \in B^+(S)$ (denoted $S' \models \varphi$) if by assigning 'true' to all members of S' and 'false' to all members of $S \setminus S'$ the formula φ evaluates to 'true'. Clearly 'true' is satisfied by the empty set and 'false' cannot be satisfied.

A *tree* is a set $T \subseteq \mathbb{N}^*$ such that if $x \cdot c \in T$ where $x \in \mathbb{N}^*$ and $c \in \mathbb{N}$, then also $x \in T$. The elements of T are called *nodes*, and the empty word ϵ is the *root* of T. For every $x \in T$, the nodes $x \cdot c$ where $c \in \mathbb{N}$ are the *successors* of x. A node is a *leaf* if it has no successors. A *path* π of a tree T is a set $\pi \subseteq T$ such that $\epsilon \in \pi$ and for every $x \in \pi$, either x is a leaf or there exists a unique $c \in \mathbb{N}$ such that $x \cdot c \in \pi$. Given an alphabet Σ, a *Σ-labeled tree* is a pair (T, V) where T is a tree and $V : T \to \Sigma$ maps each node of T to a letter in Σ.

An *1-way alternating automaton* is $B = \langle \Sigma, Q, s_0, \Delta, F \rangle$ where Σ, Q and F are like in nondeterministic automata. s_0 is a unique starting state and $\Delta : S \times \Sigma \to B^+(Q)$

is the transition function. Again we may run A on finite words (*1-way alternating automata on finite words* or *1AFA* for short) or on infinite words (*1-way alternating Büchi automata* or *1ABW* for short).

A *run* of A on a finite word $w = w_0...w_l$ is a labeled tree (T, r) where $r : T \to Q$. The maximal depth in the tree is $l + 1$. A node x labeled by s describes a copy of the automaton in state s reading letter $w_{|x|}$. The labels of a node and its successors have to satisfy the transition function Δ. Formally, $r(\epsilon) = s_0$ and for all nodes x with $r(x) = s$ and $\Delta(s, w_{|x|}) = \phi$ there is a (possibly empty) set $\{s_1, ..., s_n\} \models \phi$ such that for each state s_i there is a successor of x labeled s_i. The run is *accepting* if all the leaves in depth $l + 1$ are labeled by states from F.

A run of A on an infinite word $w = w_0 w_1...$ is defined similarly as a (possibly) infinite labeled tree. A run of a 1ABW is *accepting* if every infinite path visits the accepting set infinitely often. As before, a word w is *accepted* by A if it has an accepting run over the word. We similarly define the language of A, $L(A)$.

3 Automata on Finite Words

We start by transforming 2NFA to 1AFA. We analyze the possible form of an accepting run of a 2NFA and using a 1AFA check when such a run exists over a word.

Theorem 1. *For every 2NFA $A = \langle \Sigma, S, s_0, \rho, F\rangle$ with n states, there exists an 1AFA $B = \langle \Sigma, Q, s_0, \Delta, F\rangle$ with $O(n^2)$ states such that $L(B) = L(A)$.*

Given a 2NFA $A = \langle \Sigma, S, s_0, \delta, F\rangle$, let $B = \langle \Sigma, Q, s_0, \Delta, F\rangle$ denote its equivalent 1AFA. Note that B uses the acceptance set and the initial state of A.

Recall that a run of A is a sequence $r = (s_0, 0), (s_1, i_1), (s_2, i_2), ..., (s_m, i_m)$ of pairs of states and locations, where s_j is the state and i_j is the location of the automaton in the word w. We refer to each state as a *forward* or *backward* state according to its predecessor in the run. If it resulted from a backward movement it is a *backward* state and if from a forward movement it is a *forward* state. Formally, (s_j, i_j) is a forward state if $i_j = i_{j-1} + 1$ and backward state if $i_j = i_{j-1} - 1$. The first state $(s_0, 0)$ is defined to be a forward state.

Given the 2NFA A our goal is to construct the 1AFA B recognizing the same language. In Figure 1a we see that a run of A takes the form of a tree of 'zigzags'. Our one-way automaton reads words moving forward and accepts if such a tree exists. In Figure 1a we see that there are two transitions using a_1. The first $(s_2, 1) \in \delta(s_1, a_1)$ and the second $(s_4, 1) \in \delta(s_3, a_1)$. In the one-way sweep we would like to make sure that s_3 indeed resulted from s_2 and that the run continuing from s_3 to s_4 and further is accepting. Hence when in state s_1 reading letter a_1 we guess that there is a part of the run coming from the future and spawn two processes. The first checks that s_1 indeed results in s_3 and the second ensures that the part $s_3, s_4, ...$ of the run is accepting.

Hence the state set of the alternating automaton is $Q = S \cup (S \times S)$. A state $s \in Q$ represents a part of the run that is only looking forward (s_4 in Figure 1a). A pair state $(s_1, s_3) \in Q$ represents a part of the run that consists of a forward moving state and a backward moving state (s_1 and s_3 in Figure 1a). Such a pair ensures that there is a run segment linking the forward state to the backward state. We introduce one modification, since s_3 is a backward state (i.e. $(s_3, -1) \in \delta(s_2, a_2)$) it makes sense to associate it with a_2 and not with a_1. As the alternating automaton reads a_1 (when in state s_1), it guesses that s_3 comes from the future and changes direction. The alternating automaton then spawns two processes: the first, s_4 and the second, (s_2, s_3), and both read a_2 as their next letter. Then it is easier to check that $(s_3, -1) \in \delta(s_2, a_2)$.

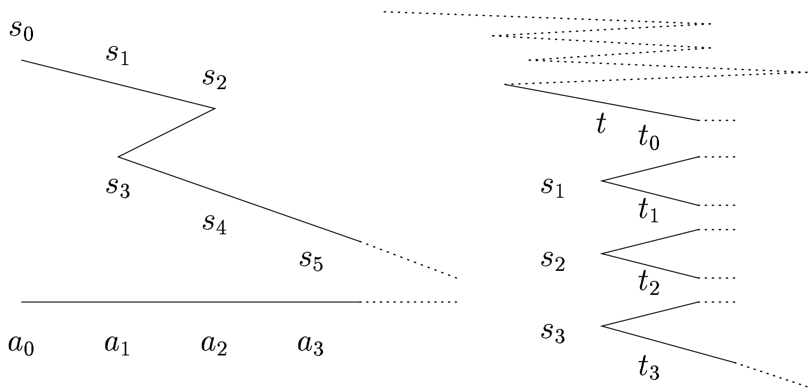

Fig. 1. (a) A zigzag run (b) The transition at the singleton state t

3.1 The Construction

The transition at a singleton state We define the transitions of B in two stages. First we define transitions from a singleton state. When in a singleton state $t \in Q$ reading letter a_j (See Figure 1b) the alternating automaton guesses that there are going to be k more visits to letter a_j in the rest of the run (as the run is simple k is bounded by the number of states of the 2NFA A, $|S| = n$). We refer to the states reading letter a_j according to the order they appear in the run as $s_1, ..., s_k$. We assume that all states that read letters prior to a_j have already been taken care of, hence $s_1, ..., s_k$ themselves are backward states (i.e. $(s_i, -1) \in \delta(p_i, a_{j+1})$ for some p_i). They read the letter a_j and move forward (there exists some t_i such that $(t_i, 1) \in \delta(s_i, a_j)$). Denote the successors of $s_1, ..., s_k$ by $t_1, ..., t_k$. The alternating automaton verifies that there is a run segment connecting the successor of t (denoted t_0) to s_1 (by induction, all states reading letters before a_j have been taken care of, this run segment should not go back to letters before a_j). Similarly verify that a run segment connects t_1 to s_2, etc. In general the automaton checks that there is a part of the run connecting t_i to s_{i+1}. Finally, from t_k the run has to go on moving forward and reach location $|w|$ in an accepting state.

Given a state t and an alphabet letter a, consider the set R^t_a of all possible sequences of states of length at most $2n - 1$ where no two states in an even place (forward states) are equal and no two states in an odd place (backward states) are equal. We further demand that the first state in the sequence be a successor of t ($(t_0, 1) \in \delta(t, a)$) and similarly that t_i be a successor of s_i ($(t_i, 1) \in \delta(s_i, a)$). Formally

$$R^t_a = \left\{ \langle t_0, s_1, t_1, ..., s_k, t_k \rangle \;\middle|\; \begin{array}{l} 0 \leq k < n \\ (t_0, 1) \in \delta(t, a) \\ \forall i < j,\ s_i \neq s_j \text{ and } t_i \neq t_j \\ \forall i,\ (t_i, 1) \in \delta(s_i, a) \end{array} \right\}$$

The transition of B chooses one of these sequences and ensures that all promises are kept, i.e. there exists a run segment connecting t_{i-1} to s_i.

$$\Delta(t, a) = \bigvee\nolimits_{\langle t_0, \ldots, t_k \rangle \in R^t_a} (t_0, s_1) \wedge (t_1, s_2) \wedge ... \wedge (t_{k-1}, s_k) \wedge t_k$$

The transition at a pair state When the alternating automaton is in a pair state (t, s) reading letter a_j it tries to find a run segment connecting t to s using only the suffix $a_j ... a_{|w|-1}$. We view t as a forward state reading a_j and s as a backward state reading a_{j-1} (Again $(s, -1) \in \delta(p, a_j)$). As shown in Figure 2a, the run segment connecting t to s might visit letter a_j but should not visit a_{j-1}.

Figure 2b provides a detailed example. The automaton in state (t, s) guesses that the run segment linking t to s visits a_2 twice and that the states reading letter a_2 are s_1 and s_2. The automaton further guesses that the predecessor of s is s_3 ($(s, -1) \in \delta(s_3, a_2)$) and that the successors of t, s_1 and s_2 are t_0, t_1 and t_2 respectively. The alternating automaton spawns three processes: (t_0, s_1), (t_1, s_2) and (t_2, s_3) all reading letter a_{j+1}. Each of these pair states has to find a run segment connecting the two states.

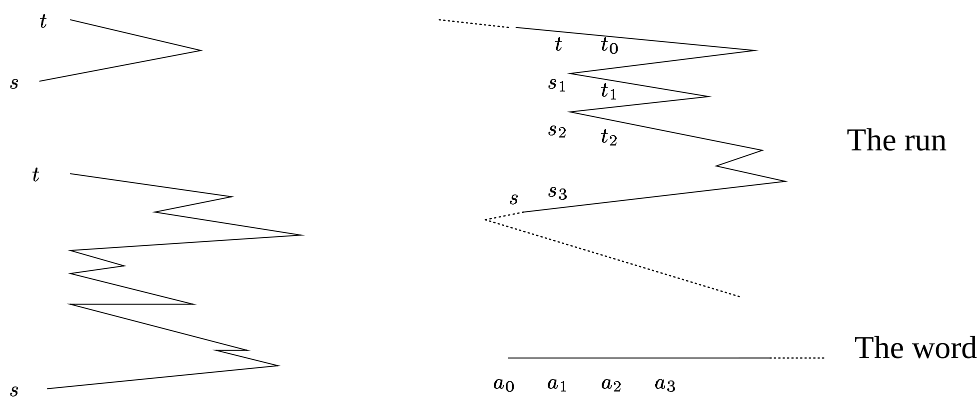

Fig. 2. (a) Different connecting segments (b) The transition at the pair state (t, s)

We now define the transition from a state in $S \times S$. Given a state (t, s) and an alphabet letter a, we define the set $R^{(t,s)}_a$ of all possible sequences of states of length at most $2n$ where no two states in an even position (forward states) are equal and no two states in an odd position (backward states) are equal. We further demand that the first state in the sequence be a successor of t ($(t_0, 1) \in \delta(t, a)$), that the last state in the sequence be a predecessor of s ($(s, -1) \in \delta(s_{k+1}, a)$) and similarly that t_i be a successor of s_i ($(t_i, 1) \in \delta(s_i, a)$).

$$R_a^{(t,s)} = \left\{ \langle t_0, s_1, t_1, ..., s_k, t_k, s_{k+1} \rangle \;\middle|\; \begin{array}{l} 0 \leq k < n \\ (t_0, 1) \in \delta(t, a) \\ (s, -1) \in \delta(s_{k+1}, a) \\ \forall i,\ (t_i, 1) \in \delta(s_i, a) \end{array} \right\}$$

The transition of B chooses one sequence and ensures that all pairs meet:

$$\Delta((t, s), a) = \begin{cases} true & \text{If } (s, -1) \in \delta(t, a) \\ \displaystyle\bigvee_{\langle t_0, \ldots, s_{k+1} \rangle \in R_a^{(t,s)}} (t_0, s_1) \wedge (t_1, s_2) \wedge ... \wedge (t_k, s_{k+1}) & \text{Otherwise} \end{cases}$$

Claim. $L(A) = L(B)$

Proof. Given an accepting simple run of A on a word w of the form $(s_0, 0)$, (s_1, i_1), $\ldots, (s_m, i_m)$, we annotate each pair by the place it took in the run of A. Thus the run takes the form $(s_0, 0, 0)$, $(s_1, i_1, 1), \ldots, (s_m, i_m, m)$. We build a run tree (T, V) of B by induction. In addition to the labeling $V : T \rightarrow S \cup S \times S$, we attach a single tag to a singleton state and a pair of tags to a pair state. The tags are triplets from the annotated run of A. For example the root of the run tree of B is labeled by s_0 and tagged by $(s_0, 0, 0)$. The labeling and the tagging conforms to the following:

- Given a node x labeled by state s tagged by (s', i, j) from the run of A we build the tree so that $s = s',\ i = |x|$ and furthermore all triplets in the run of A whose third element is larger than j have their second element at least i.
- Given a node x labeled by state (t, s) tagged by (t', i_1, j_1) and (s', i_2, j_2) in the run of A we build the tree so that $t = t',\ s = s',\ i_1 = i_2 + 1 = |x|$, $j_1 < j_2$ and that all triplets in the run of A whose third element is between j_1 and j_2 have their second element be at least i_1.

We start with the root labeling it by s_0 and tagging it by $(s_0, 0, 0)$. Obviously this conforms to our demands.

Given a node x labeled by s tagged by (s, i, j) adhering to our demands (see state t in Figure 1b). If (s, i, j) has no successor in the run of A, it must be the case that $i = |w|$ and that $s \in F$. Otherwise we denote the triplets in the run of A whose third element is larger than j and whose second element is i by $(s_1, i, j_1), ..., (s_k, i, j_k)$. By assumption there is no point in the run of A beyond j visiting a letter before i. Since the run is simple $k < n$. Denote by $(t_0, i + 1, j + 1)$ the successor of (s, i, j) and by $(t_1, i + 1, j_1 + 1), ..., (t_k, i + 1, j_k + 1)$ the successors of $s_1, ..., s_k$. We add $k + 1$ successors to x, label them $(t_0, s_1), (t_1, s_2), ..., (t_{k-1}, s_k), t_k$ and tag them in the obvious way. We show now that the new nodes added to the tree conform to our demands. By assumption there are no visits beyond the j^{th} step in the run of A to letters before a_i and $s_1, ..., s_k$ are all the visits to a_i after the j^{th} step of A.

Let $y = x \cdot c$ be the successor of x labeled t_k (tagged $(t_k, i+1, j_k+1)$). Since $|x| = i$, we conclude $|y| = i+1$. All the triplets in the run of A appearing after $(t_k, i+1, j_k+1)$ do not visit letters before a_{i+1} (We collected all visits to a_i).

Let $y = x \cdot d$ be a successor of x labeled by (t_l, s_{l+1}) (tagged $(t_l, i+1, j_l+1)$ and (s_{l+1}, i, j_{l+1})). We know that $i = |x|$ hence $i+1 = |y|$, $j_l + 1 < j_{l+1}$ and between the $j_l + 1$ element in the run of A and the j_{l+1} element letters before a_{i+1} are not visited.

We turn to continuing the tree below a node labeled by a pair state. Given a node x labeled by (t, s) tagged (t, i, j) and and $(s, i-1, k)$. By assumption there are no visits to a_{i-1} in the run of A between the j^{th}triplet and k^{th} triplet. If $k = j+1$ then we are done and we leave this node as a leaf. Otherwise we denote the triplets in the run of A whose third element is between j and k and whose second element is i by $s_1, ..., s_k$ (see Figure 2b). Denote by $t_1, ..., t_k$ their successors, by t_0 the successor of t and by s_{k+1} the predecessor of s. We add $k+1$ successors to x and label them $(t_0, s_1), (t_1, s_2), ..., (t_k, s_{k+1})$, tagging is obvious. As in the previous case when we combine the assumption with the way we chose $t_0, ...t_k$ and $s_1, ..., s_{k+1}$, we conclude that the new nodes conform to the demands.

Clearly, all pair-labeled paths terminate with 'true' before reading the whole word w and the path labeled by singleton states reaches the end of w with an accepting state.

In the other direction we stretch the tree run of B into a linear run of A. In Appendix C we give a recursive algorithm that starts from the root of the run tree and constructs a run of A. When first reaching a node x labeled by pair-state (s, t), we add s to the run of A. Then we handle recursively the sons of x. When we return to x we add t to the run of A. When reaching a node x labeled by a singleton state s we simply add s to the run of A and handle the sons of x recursively. $\square$

4 Automata on infinite words

We may try to run the 1AFA from Section 3 on infinite words. We demand that pair-labeled paths be finite and that the infinite singleton-labeled path visit F infinitely often. Although an accepting run of A visited F infinitely often we cannot ensure infinitely many visits to F on the infinite path. The visits may be reflected in the run of B in the pair-labeled paths. Another problem is when the run ends in a loop.

Theorem 2. *For every 2NBW $A = \langle \Sigma, S, s_0, \rho, F \rangle$ with n states, there exists an 1ABWs $B = \langle \Sigma, Q, s'_0, \Delta, F' \rangle$ with $O(n^2)$ states such that $L(B) = L(A)$.*

We have to record hidden visits to F. This is done by doubling the set of states. While in the finite case the state set is $S \cup S \times S$, this time we also annotate the states by $\perp$ and $\top$. Hence $Q = (S \cup S \times S) \times \{\perp, \top\}$. A pair state labeled by $\top$ is a promise to visit the acceptance set. The state $(s, t, \top)$ means that in the run segment linking s to t there has to appear a state from F. A state $(s, \top)$ is displaying a visit to F in the zigzags connecting s to the previous singleton state. The initial state is $s'_0 = (s_0, \perp)$.

With the same notation we solve the problem of a loop. We allow a transition from a singleton state to a sequence of pair states. One of the pairs promises a visit to F. The acceptance set is $F' = (S \times \{\top\})$ and the transition function Δ is defined as follows.

The transition at a singleton state Just like in the finite case we consider all possible sequences of states of length at most $2n-1$ with same demands.

$$R_a^t = \left\{ \langle t_0, s_1, t_1, ..., s_k, t_k \rangle \;\middle|\; \begin{array}{l} 0 \le k < n \\ (t_0, 1) \in \delta(t, a) \\ \forall i < j,\ s_i \ne s_j \text{ and } t_i \ne t_j \\ \forall i,\ (t_i, 1) \in \delta(s_i, a) \end{array} \right\}$$

Recall that a sequence $(t_0, s_1), (t_1, s_2), ..., (t_{k-1}, s_k), t_k$ checks that there is a zigzag run segment linking t_0 to t_k. We mentioned that t_k is annotated with $\top$ in case this run segment has a visit to F. If t_k is annotated with $\top$, at least one of the pairs has to be annotated with $\top$. Although more than one pair might visit F we annotate all other pairs by $\perp$. Hence for a sequence $\langle t_0, s_1, t_1, ..., s_k, t_k \rangle$ we consider the sequences of $\perp$ and $\top$ of length $k+1$ in which if the last is $\top$ so is another one. Otherwise all are $\perp$.

$$\alpha_k^R = \left\{ \langle \alpha_0, ..., \alpha_k \rangle \in \{\perp, \top\}^{k+1} \;\middle|\; \begin{array}{l} \text{If } \alpha_k = \top \text{ then } \exists! i \text{ s.t. } 0 \le i < k \text{ and } \alpha_i = \top \\ \text{If } \alpha_k = \perp \text{ then } \forall\, 0 \le i < k,\ \alpha_i = \perp \end{array} \right\}$$

However this is not enough. We have to consider also the case of a loop. The automaton has to guess that the run terminates with a loop when it reads the first letter of w that is read inside the loop. The only states reading this letter inside the loop are backward states. We consider all sequences of at most $2n$ states and a location p within the sequence. In order to close the loop we demand either that the last backward state be equal to some previous backward state or that some forward state be a successor of the last backward state. The location p denotes the place where the loop closes ($s_{k+1} = s_p$ or $(t_p, 1) \in \delta(s_{k+1}, a)$). Sequences of length $2n$ suffice, the longest possible sequence without repetition is of length n, we may use the current state as the $n+1^{th}$ backward state or transition into one of the forward states thus creating a sequence of length $n+1$. Hence no two states in an even/odd position (forward/backward state) are equal except the last backward state. We demand that the first state in the sequence be a successor of t ($(t_0, 1) \in \delta(t, a)$), that t_i be a successor of s_i ($(t_i, 1) \in \delta(s_i, a)$) and that the p^{th} backward state be equal to the last backward state or the p^{th} forward state be a successor of the last backward state (We identify t with s_0, $s_p = s_{k+1}$ or $(t_p, 1) \in \delta(s_{k+1}, a)$).

$$L_a^t = \left\{ (\langle t_0, s_1, t_1, ..., s_k, t_k, s_{k+1} \rangle, p) \;\middle|\; \begin{array}{l} 0 \le k < n,\ 0 \le p \le k \\ (t_0, 1) \in \delta(t, a) \\ \forall i < j \ne k+1,\ s_i \ne s_j \text{ and } t_i \ne t_j \\ \forall i,\ (t_i, 1) \in \delta(s_i, a) \\ \text{if we define } s_0 = t \text{ then} \\ s_{k+1} = s_p \text{ or } (t_p, 1) \in \delta(s_{k+1}, a) \end{array} \right\}$$

It is obvious that a visit to F has to occur within the loop. Hence given the sequence $\langle t_0, s_1, t_1, ..., s_k, t_k, s_{k+1} \rangle$ and the location p we have to make sure that the run segment connecting one of the pairs between the p^{th} pair and the last pair visits F. Hence we annotate one of the pairs $(t_p, s_{p+1}), ..., (t_k, s_{k+1})$ with $\top$. In case $s_{k+1} = t$ then one of the pairs has to be annotated by $\top$. Our notation using $p = 0$ works in this case. One visit to F is enough hence all other pairs are annotated by $\perp$.

$$\alpha_{k,p}^L = \left\{ \langle \alpha_0, ..., \alpha_k \rangle \in \{\perp, \top\}^{k+1} \;\middle|\; \begin{array}{l} \forall\, 0 \le i < p,\ \alpha_i = \perp \text{ and} \\ \exists! i \text{ s.t. } \alpha_i = \top \end{array} \right\}$$

The transition of B chooses a sequence in $R_a^t \cup L_a^t$ and a sequence of $\perp$ and $\top$.

$$\Delta((t,\perp),a) = \Delta((t,\top),a) = \bigvee \begin{array}{l} \displaystyle\bigvee_{R_a^t,\alpha_k^R} (t_0,s_1,\alpha_0) \wedge ... \wedge (t_{k-1},s_k,\alpha_{k-1}) \wedge (t_k,\alpha_k) \\ \displaystyle\bigvee_{L_a^t,\alpha_{k,p}^L} (t_0,s_1,\alpha_0) \wedge ... \wedge (t_k,s_{k+1},\alpha_k) \end{array}$$

The transition at a pair state In this case the only difference is the addition of $\perp$ and $\top$. The set $R_a^{(t,s)}$ is equal to the finite case.

$$R_a^{(t,s)} = \left\{ \langle t_0,s_1,t_1,...,s_k,t_k,s_{k+1}\rangle \left| \begin{array}{l} 0 \le k < n \\ (t_0,1) \in \delta(t,a) \\ (s,-1) \in \delta(s_{k+1},a) \\ \forall i,\ (t_i,1) \in \delta(s_i,a) \end{array} \right. \right\}$$

In the transition of ‘top’ states we have to make sure that a visit to F indeed occurs. If the visit occured in this stage the promise ($\top$) can be removed ($\perp$). Otherwise the promise must be passed to one of the successors.

$$\alpha_{s,t,k}^R = \left\{ \langle \alpha_0,...,\alpha_k\rangle \in \{\perp,\top\}^{k+1} \left| \begin{array}{l} \text{If } s \notin F \text{ and } t \notin F \text{ then } \exists! i \text{ s.t. } \alpha_i = \top \\ \text{Otherwise } \forall\ 0 \le i \le k,\ \alpha_i = \perp \end{array} \right. \right\}$$

The transition of B chooses a sequence of states and a sequence of $\perp$ and $\top$.

$$\Delta((t,s,\perp),a) = \begin{cases} true & \text{If } (s,-1) \in \delta(t,a) \\ \displaystyle\bigvee_{R_a^{(t,s)}} (t_0,s_1,\perp) \wedge ... \wedge (t_k,s_{k+1},\perp) & \text{Otherwise} \end{cases}$$

$$\Delta((t,s,\top),a) = \begin{cases} true & \text{If } (s,-1) \in \delta(t,a) \text{ and} \\ & (s \in F \text{ or } t \in F) \\ \displaystyle\bigvee_{R_a^{(t,s)},\alpha_{s,t,k}^R} (t_0,s_1,\alpha_0) \wedge ... \wedge (t_k,s_{k+1},\alpha_k) & \text{Otherwise} \end{cases}$$

Claim. L(A)=L(B)

The proof is just an elaboration on the proof of the finite case. In Appendix D we hilight the points of difference.

Remark: In both the finite and the infinite cases, we get a 1-way alternating automaton with $O(n^2)$ states and transitions of exponential size. Birget's construction also results in exponential-sized transitions [Bir93]. Globerman and Harel use 0-steps in order to reduce the transition to polynomial size [GH96]. Their construction uses the reverse language and can not be applied to infinite words. If we use 0-steps, it is quite simple to change our construction so that it uses only polynomial-sized transitions. We note that the transition size does not effect the conversion from 1ABW to 1NBW.

5 Acknowledgments

We would like to thank Orna Kupferman for her remarks on the manuscript.

References

[Bir93] J.C. Birget. State-complexity of finite-state devices, state compressibility and incompressibility. *Mathematical Systems Theory*, 26(3):237–269, 1993.

[BL80] J.A. Brzozowski and E. Leiss. Finite automata and sequential networks. *Theoretical Computer Science*, 10:19–35, 1980.

[Büc62] J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Internat. Congr. Logic, Method. and Philos. Sci. 1960*, pages 1–12, Stanford, 1962. Stanford University Press.

[CdGLV00] D. Calvanese, G. de Giacomo, M. Lenzerini, and M.Y. Vardi. View-based query processing for regular path queries with inverse. In *Proc. ACM 19th Symposium on Principles of Database Systems*, pages 58–66, 2000.

[CKS81] A.K. Chandra, D.C. Kozen, and L.J. Stockmeyer. Alternation. *Journal of the Association for Computing Machinery*, 28(1):114–133, January 1981.

[GH96] N. Globerman and D. Harel. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science*, 143:161–184, 1996.

[HK96] G. Holzmann and O. Kupferman. Not checking for closure under stuttering. In *The Spin Verifi cation System*, pages 17–22. American Mathematical Society, 1996. Proc. 2nd International SPIN Workshop.

[JPAR68] W.L. Johnson, J.H. Porter, S.I. Ackley, and D.T. Ross. Automatic generatin of efficient lexical processors using finite state techniques. *Communications of the ACM*, 11(12):805–813, 1968.

[Koh70] Z. Kohavi. *Switching and Finite Automata Theory*. McGraw-Hill, New York, 1970.

[Kur94] R.P. Kurshan. *Computer Aided Verifi cation of Coordinating Processes*. Princeton Univ. Press, 1994.

[KV00] O. Kupferman and M.Y. Vardi. Synthesis with incomplete informatio. In *Advances in Temporal Logic*, pages 109–127. Kluwer Academic Publishers, January 2000.

[LLS84] Richard E. Ladner, Richard J. Lipton, and Larry J. Stockmeyer. Alternating pushdown and stack automata. *SIAM Journal on Computing*, 13(1):135–155, February 1984.

[McN66] R. McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[Rab69] M.O. Rabin. Decidability of second order theories and automata on infinite trees. *Transaction of the AMS*, 141:1–35, 1969.

[RS59] M.O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3:115–125, 1959.

[She59] J. C. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3:198–200, 1959.

[SS78] W. J. Sakoda and M. Sipser. Nondeterminism and the size of two way finite automata. In *Tenth Annual ACM Symposium on Theory of Computing*, pages 275–286, San Diego, California, May 1978. ACM.

[Str82] R.S. Streett. Propositional dynamic logic of looping and converse. *Information and Control*, 54:121–141, 1982.

[Var88] M.Y. Vardi. A temporal fixpoint calculus. In *Proc. 15th ACM Symp. on Principles of Programming Languages*, pages 250–259, San Diego, January 1988.

[Var89] Moshe Y. Vardi. A note on the reduction of two-way automata to one-way automata. *Information Processing Letters*, 30(5):261–264, March 1989.

[Var90] Moshe Y. Vardi. Endmarkers can make a difference. *Information Processing Letters*, 35(3):145–148, July 1990.

[Var98] M.Y. Vardi. Reasoning about the past with two-way automata. In *Proc. 25th International Coll. on Automata, Languages, and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer-Verlag, Berlin, July 1998.

[VW86] M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st Symp. on Logic in Computer Science*, pages 332–344, Cambridge, June 1986.

[Wil99] T. Wilke. CTL$^+$ is exponentially more succinct than CTL. In C. Pandu Ragan, V. Raman, and R. Ramanujam, editors, *Proc. 19th conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer-Verlag, 1999.

A Always moving automata

In this section we show that every 2-way nondeterministic automaton can be converted to an automaton whose transition is of the form $\rho : S \times \Sigma \to 2^{S\times\{-1,1\}}$.

Given a 2-way automaton $A = \langle \Sigma, S, s_0, \rho, F\rangle$, a *0-step* in a run of A is when two adjacent states in the run read the same letter. Formally, in the run $(s_0, i_0), (s_1, i_1), ..., (s_m, i_m), ...,$ step $j > 0$ is a 0-step if $i_j = i_{j-1}$.

A.1 Automata on finite words

Given a 2NFA $A = \langle \Sigma, S, s_0, \delta, F\rangle$ with $\delta : S \times \Sigma \to 2^{S\times\{-1,0,1\}}$ we construct $A' = \langle \Sigma, S, s_0, \delta', F\rangle$ with $\delta' : S' \times \Sigma \to 2^{S\times\{-1,1\}}$ (i.e. $L(A) = L(A')$). There are no 0-steps in the run of the second.

For each state s and alphabet letter a, the set C^s_a of all states reachable from s with 0 steps using letter a. We call C^s_a the *0-closure* of s and a.

$$C^s_a = \{t \in S | \exists s_1, ..., s_k \text{ s.t. } 1 \le k,\ s_1 = s,\ s_k = t \text{ and } (s_{i+1}, 0) \in \delta(s_i, a)\}$$

Define $\delta'(s, a) = \bigcup_{t\in C^s_a} \delta(t, a)$ and take $\delta' = \delta'' \cap (S \times \{-1, 1\})$ (i.e. remove all pairs of the form $S \times \{0\}$). This way the closure takes care of the 0-steps and A' takes steps either forward or backward.

Claim. $L(A) = L(A')$

Proof. Suppose A accepts w. Let $r = (s_0, 0), ..., (s_m, i_m)$ be an accepting run of A on w. We turn r into a run r' of A' on w by pruning 0-steps: if $i_j = i_{j-1}$ simply remove (s_j, i_j) from the run. It is easy to see that r' is an accepting run of A' on w.

Suppose A' accepts w. Let $r' = (s_0, 0), ..., (s_m, i_m)$ be an accepting run of A' on w. We append the 0-steps from the closure of each state to complete a run of A on w. □

A.2 Automata on infinite words

In the infinite case there are two potential complications. Visits to F in a 0-step and a loop of 0-steps that visits F. In order to solve these problems, we double the number of states and add an accepting sink state. (in [Wil99,HK96] similar problems are solved in a similar way).

Given the 2NBW $A = \langle \Sigma, S, s_0, \delta, F\rangle$ where $\delta : S \times \Sigma \to 2^{S\times\{-1,0,1\}}$ we show that the automaton $A' = \langle \Sigma, (S \times \{\perp, \top\}) \cup \{Acc\}, (s_0, \perp), \delta', (S \times \{\top\}) \cup \{Acc\}\rangle$ accepts the same language. Furthermore, A' is 0-step free.

Given a state s and an alphabet letter a, we define NC_a^s the set of all states reachable from state s by a sequence of 0-steps reading letter a and one last forward/backward step. All states avoid the acceptance set F.

$$NC_a^s = \left\{ ((t,\perp), i) \in ((S \times \{\perp\}) \times \{-1,1\}) \;\middle|\; \begin{array}{l} \exists(s_0, ..., s_k) \in \{s\} \cdot (S \setminus F)^k \\ s.t.\ 1 \le k,\ s_0 = s,\ s_k = t, \\ \forall 0 \le j < k,\ (s_{j+1}, 0) \in \delta(s_j, a) \\ and\ (s_k, i) \in \delta(s_{k-1}, a) \end{array} \right\}$$

In addition we define AC_a^s the set of all states reachable from state s by a sequence of 0-steps reading letter a and one last forward/backward step. One of the states in the sequence is an accepting state.

$$AC_a^s = \left\{ ((t,\top), i) \in ((S \times \{\top\}) \times \{-1,1\}) \;\middle|\; \begin{array}{l} \exists(s_0, ..., s_k) \in \{s\} \cdot S^k\ s.t.\ 1 \le k, \\ s_0 = s,\ s_k = t,\ \exists j > 0\ s.t.\ s_j \in F, \\ \forall 0 \le j < k,\ (s_{j+1}, 0) \in \delta(s_j, a) \\ and\ (s_k, i) \in \delta(s_{k-1}, a) \end{array} \right\}$$

We also have to take care of situations where there is a loop of 0-steps that visits F. The boolean variable $ACCEPT_a^s$ is set to 1 if such a sequence exists and to 0 otherwise. Formally, the variable $ACCEPT_a^s$ is set to 1 iff there exists a sequence $(s_0, ..., s_k) \in \{s\} \cdot S^k$, where $1 \le k$ and all the following conditions hold.

- $s_0 = s$.
- There exist j and l such that $0 \le j \le l < k$, $s_k = s_j$ and $s_l \in F$.
- For all j where $0 \le j \le k$, we have $(s_{j+1}, 0) \in \delta(s_j, a)$

We use the two 0-closures and the variable defined above in the definition of the transition function of the 1AFA B.

$$\delta'((s,\perp), a) = \delta'((s,\top), a) = \begin{cases} \{(Acc, 1)\} & ACCEPT_a^s = 1 \\ NC_a^s \cup AC_a^s & ACCEPT_a^s = 0 \end{cases}$$
$$\delta'(Acc, a) = \{(Acc, 1)\}$$

Apparently, A' is 0-step free.

Claim. L(A')=L(A)

Proof. Suppose A accepts w. There exists an accepting run r of A on w. If a finite sequence of 0-steps appears in r we simply prune it. If that sequence contained a visit to F

add $\top$ to the forward/backward move at the end of the sequence. If r ends in an infinite sequence of 0-steps, this sequence has a finite prefix $(s_i, l), (s_{i+1}, l), ..., (s_{i+p}, l)$ such that $s_i = s_{i+p}$ and, as r is accepting, there is a visit to F in this prefix. We take the prefix of the run $(s_0, 0), ..., (s_i, l)$ and add to it the infinite suffix $(Acc, l+1), (Acc, l+2), ...$. Finally, we add labels $\perp$ to all unlabeled states. It is easy to see that the resulting run is a valid run of A'. It is also an accepting run. If the run ends in a suffix Acc^ω then it is clearly accepting. Otherwise, removing sequences of 0-steps replaces a finite number of visits to F by a state labeled by $\top$. As the original run visited F infinitely often, so does the run of A'.

Suppose A' accepts w. We append 0-steps as promised from the definition of NC and AC. If the run ends with an infinite sequence of Acc we can add a loop visiting F. Infinitely many occurrences of $\top$ ensure infinitely many visits to F. □

B Simple runs are enough

Given a 2NFA/2NBW $A = \langle \Sigma, S, s_0, \rho, F \rangle$ we claim the following.

Claim. The automaton A accepts a word w iff it accepts it with a simple run.

Proof (The finite case). A simple run is a run. Given an accepting run $r = (s_0, 0), (s_1, i_1), (s_2, i_2), \ldots, (s_m, i_m)$ of A on w, we construct a simple run of A on w. If r is not simple, there are some j and k such that $j < k$, $s_j = s_k$ and $i_j = i_k$, consider the sequence $(s_0, 0), \ldots, (s_j, i_j), (s_{k_1}, i_{k+1}), \ldots, (s_m, i_m)$. Since $(s_{k+1}, i_{k+1} - i_k) \in \delta(s_k, a_{i_k})$ and $\delta(s_k, a_{i_k}) = \delta(s_j, a_{i_j})$ this sequence is still a run. The last state s_m is a member of F and $i_m = |w|$ hence the run is accepting. Since the run is finite, finitely many repetitions of the above operation result in a simple run of A on w. □

Proof (The infinite case). A simple run is a run. Given a run $r = (s_0, 0), (s_1, i_1), (s_2, i_2), \ldots$, we cannot simply remove sequences of states like we did in the finite case, the visits to F might be hidden in these parts of the run. If for some $j < k$, we have that $s_j = s_k$, $i_j = i_k$ and $s_p \notin F$ for all $j \leq p \leq k$, we can simply remove this part. As in the finite case, the run stays a valid accepting run.Now if there exists some $j < k$ such that $s_j = s_k$ and $i_j = i_k$ we conclude that there is a visit to F between the two. We take the minimal j and k and create the run $(s_0, 0), \ldots, (s_{j-1}, i_{j-1}), ((s_j, i_j), \ldots, (s_{k-1}, i_{k-1}))^\omega$. Again this is a valid run and it visits F infinitely often (between s_j and s_{k-1}). If no such j and k exist the run is simple. □

C Proof of correctness of construction in the finite case

Proof. Given an accepting run tree of B on a word w, we turn it into a linear run of A. We assume ordering on the successors of each node according to the appearance of their labels in the sets R_a. We give a recursive algorithm to build the run of A.

Starting from the root ϵ labeled $(s_0, 0)$, we add to the run of A the element $(s_0, 0)$. We now handle the successors of the root according to their order. Going up to the first successor c labeled (t, s) we add $(t, 1)$ to the run of A. Obviously from the definition of $R_{a_0}^{s_0}$ we know that $(t, 1) \in \delta(s_0, a_0)$. We handle the successors of c in the recursion.

When we return to c we add $(s, 0)$ to the run of A (to be justified later). We return now to ϵ and handle the next successor d. The node d is either labeled by (p, q) or by p. In both cases the definition of $R^{s_0}_{a_0}$ ensures that $(p, 1) \in \delta(s, a_0)$. When we return to ϵ after scanning the whole tree the run of A is complete.

Getting to a node x labeled (t, s) we add $(t, |x|)$ to the run of x. Adding $(t, |x|)$ itself and passing to the successors of x and between them was justified when handling the root. When the recursion finished handling the last successor of x we add $(s, |x| - 1)$ to the run of A. Suppose the last successor of x was labeled (p, q) then from the definition of $R^{(t,s)}_{a_{|x|}}$ we know that $(s, -1) \in \delta(q, a_{|x|})$ hence this transition is justified.

Getting to a node x labeled s is not different from handling the root. Instead of using the locations 0 and 1 in the run, we use locations $|x|$ and $|x| + 1$.

We have to show that the run is valid and accepting. Satisfying the transition was shown. In the tree run of B there is a single path labeled solely by single states. The last element in the run of A is the same state and reading the same letter as the last in this path. Since the path is accepting the last state there has to be from F and reading letter $|w|$ (which does not exists, $w = a_0 ... a_{|w|-1}$). All other triplets in the run of A read letters in the range $\{0, ..., |w| - 1\}$. Otherwise there is some node x in the run of B such that $|x| \geq |w|$ (other than the previously designated node). This is impossible since the run of B is accepting. □

D Proof of correctness of the construction for the infinite case

Given the 2-way nondeterministic Büchi automaton $A = \langle \Sigma, S, s_0, \delta, F \rangle$ we constructed in Section 4 the 1-way alternating automaton $B = \langle \Sigma, Q, s'_0, \Delta, F' \rangle$ where $Q = (S \cup S \times S) \times \{\bot, \top\}$, $F' = (S \times \{\top\})$ and the transition function Δ as defined there.

Claim. L(A)=L(B)

Proof. Given an accepting simple run of A on a word w of the form $(s_0, 0), (s_1, i_1), ...$ we annotate each pair by the place it took in the run of A. Thus the run takes the form $(s_0, 0, 0), (s_1, i_1, 1), ...$ If the run does not end in a loop the construction in the finite case works. We have to add the symbols $\bot$ and $\top$.

When dealing with a node x in the run tree of B labeled by (s, α) tagged by (s, i, j). In the proof of the finite case we identified the triplets (s_1, i, j_1), ..., (s_k, i, j_k) and $(t_0, i + 1, j + 1)$, ..., $(t_k, i + 1, j_k + 1)$ and labeled the successors of x with (t_0, s_1), ..., $(t_{k-1}, s_k), t_k$. If there is no visit to F between $j + 1$ and $j_k + 1$ we add to these states $\bot$. Otherwise the visit was between $j_l + 1$ and j_{l+1} for some l (consider $j = j_0$), in this case we add $\top$ both to t_k and to the pair (t_l, s_{l+1}), to all other pairs we add $\bot$.

When dealing with a node x in the run tree of B labeled by (t, s, α) tagged (t, i, j) and $(s, i - 1, k)$. We identified the set of pairs (t_0, s_1), ..., (t_k, s_{k+1}). In case $\alpha = \bot$ we continue just like in the finite case. In case $\alpha = \top$ we put it there because there was a visit to F between j and k. This visit to F has to occur between t_l and s_{l+1} for some l and we pass the obligation to this pair. At some point we reach a visit to F and then the promise is removed.

We have now an infinite run tree of B. All pair-labeled paths are still finite and there is one infinite path labeled by singleton states. Since every occurrence of $\top$ on this path covers a finite number of visits to F we are ensured that $\top$ appears infinitely often along this path.

If the run ends in a loop we have to identify the first letter of w read in this loop. Suppose this letter is i. We build the run tree of B as usual until reaching the node x in level i labeled by a singleton state (s, α). As letter i is visited in the loop there are infinitely many visits to it. Denote these visits by $(s_1, i, j_1), (s_2, i, j_2), \ldots$, all backward states. Denote $s = s_0$, and the successors of $s_0, ..., s_n$ by $t_0, ..., t_n$. Since the sequence $s_0, ..., s_n$ is $n+1$ long, it has to include the same state occuring twice. Denote its second occurrence by s_m. We consider two cases:

- In case t_{m-1} appears twice in the sequence $t_0, ..., t_n$ before location $m-1$, i.e. $t_{m-1} = t_p$ where $p < m-1$. In this case denote $k+1 = m-1$ and take $t_0, s_1, t_1, s_2, ..., t_{m-2}, s_{m-1}$ as the sequence from $L^t_{a_{|x|}}$ $((t_p, 1) = (t_k, 1) \in \delta(s_k, a_{|x|}))$.
- Otherwise we denote $k+1 = m$ and take $t_0, s_1, t_1, s_2, ..., t_{m-1}k, s_{k+1}$ as the sequence from $L^t_{a_{|x|}}$. Since s_{k+1} was the second occurrence there is a first occurrence $s_p = s_{k+1}$.

Since the run is simple its suffix is of the form:

$$(s_p, i), ((t_p, i+1), ..., (s_{p+1}, i), (t_{p+1}, i+1), ..., (s_k, i), (t_k, i+1), ..., (s_{k+1}, i))^\omega$$

One of the segments $(t_l, i+1), ..., (s_{l+1}, i)$ visits F. Annotate the pair (t_l, s_{l+1}) by $\top$ and all the others by $\perp$.

In the other direction we apply the same recursive algorithm. If the accepting run tree of B is infinite then we never return to ϵ but the run created is an accepting run of A.

If the accepting run tree of B is finite we have to identify the point in the tree x labeled by a singleton state (s, α) under which there are no successors labeled by singleton states. In this point we identify the loop. The last successor of x is labeled (t', s', β). We know that either $s' = s$ or there is another successor of x labeled by (t'', s'', β) such that either $s'' = s'$ (in this case (t'', s'', β) is not part of the loop) or $(t'', 1) \in \delta(s', a_{|x|})$ (in this case (t'', s'', β) is part of the loop). If $s' = s$ then we put aside the run of A built so far, denote it by r. Otherwise we start handling the successors of x until taking care of all successors that do not take part in the loop. Again we put this run aside and call it r. Now we build a new run starting from the point we stopped, since the run of B is finite the recursion ends and we are left with the run r'. Our final step is to present $r(r')^\omega$ as the new run of A. Note that the run $r(r')^\omega$ is not necessarily simple. □