From Bidirectionality to Alternation

Nir Piterman*
Weizmann Institute of Science

Moshe Y. Vardi†
Rice University

August 22, 2001

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].

*Department of Computer Science and Applied Mathematics, Weizmann institute, Rehovot 76100, Israel. Email: nirp@wisdom.weizmann.ac.il

†Address: Department of Computer Science, Rice University, Houston TX 77005-1892, U.S.A. Email: vardi@cs.rice.edu

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 efficient 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 specifications in μ-calculus on context-free and prefix-recognizable systems has been reduced to questions about 2-way automata [KV00]. In a different field 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 infinite words. The theory of finite automata on infinite 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 infinite words have shown to be useful in computer-aided verification [Kur94, VW86]. We note that bidirectionality does not add expressive power also in the context of automata on infinite 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 efficiently 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" [1]. 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. 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].

We also show how to obtain, still with quadratic blow-up, a 1-way alternating automaton for the complementary language. This is trivial for automata on finite words, but not for automata on infinite words. Finally, we show how to use our construction for 2-way nondeterministic Rabin and parity automata, avoiding an unnecessary blow up that results from first converting those into 2-way nondeterministic Büchi automata.

[1] The analysis of the form of the "zigzags" is similar to the analysis of runs of pushdown-automata done in [Ruz80, Ven91].

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 ith letter of the word w. The *length* of w is denoted by $|w|$ and is defined to be ω for infinite words.

A *2-way nondeterministic automaton* is $N = \langle \Sigma, S, s_0, \delta, F \rangle$, where Σ is the finite alphabet, S is the finite set of states, $s_0 \in S$ is the initial state, $\delta : S \times \Sigma \to 2^{S \times \{-1,0,1\}}$ is the transition function, and F is the acceptance set. We can run N 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).

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, $q_0 = s_0$, $i_0 = 0$, 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 N if it has an accepting run over w. The *language* of N is the set of words accepted by N, denoted by $L(N)$.

A *2-way nondeterministic parity (Rabin) automaton* (*2NPW* and *2NRW* for short) is $N = \langle \Sigma, S, s_0, \delta, \alpha \rangle$ where Σ, S, s_0 and δ are like before and $\alpha = \{F_1, ..., F_m\}$ is a partition of S ($\alpha = \{\langle G_1, B_1 \rangle, ..., \langle G_m, B_m \rangle\}$ is a subset of $2^S \times 2^S$). The *index* of the automaton is the number of sets (pairs) in its acceptance condition. A run of a 2NPW or a 2NRW is just like a run of a 2NBW. A run r of a 2NPW is accepting if the minimal index $1 \leq i \leq m$ such that r visits $F_i \times \mathbb{N}$ infinitely is even. A run r of a 2NRW is accepting if there exists an i, $1 \leq i \leq m$ such that r visits $G_i \times \mathbb{N}$ infinitely often and $B_i \times \mathbb{N}$ only finitely often.

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 $h < p < l + m$, either $s_h \neq s_p$ or $i_h \neq i_p$, and for all $f \geq l$, $s_f = s_{f+m}$ and $i_f = i_{f+m}$.

We show that there exists an accepting run iff there exists a simple accepting run.

Claim 2.1 *An automaton N (either $2NFA$, $2NBW$, $2NPW$, or $2NRW$) accepts a word w iff it accepts it with a simple run.*

Proof: For all automata, a simple run is in particular a run. Given an accepting run $r = (s_0, 0), (s_1, i_1), \ldots$ of A on w, we construct a simple run of N on w.

- For the case that N is a 2NFA. The run r is finite and ends in some pair (s_m, i_m). 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 $(s_0, 0), ..., (s_j, i_j), (s_{k+1}, i_{k+1}), ..., (s_m, i_m)$. 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.

- For the case the N is a 2NBW. We cannot simply remove sequences of states like we did in the finite case, the visits to F may 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 for all $j \leq p \leq k$ we have $s_p \notin F$ (no accepting state occurring), we can simply remove this part. We have to show that the limit of all these changes stays a valid and accepting run.

  Note that we change the run only between occurrences of states from F. So we can divide the run into segments. In each segment the first state is from F and no states from F occur elsewhere. As states from F occur infinitely often in the run we have infinitely many segments. Each of these segments is changed a finite number of times, the first state of the segment does not change nor does the last state of the segment. Gluing the segments together for a run after performing the changes results in a valid run (the first and the last state in every segment do not change). As every segment starts with a state from F and there are infinitely many segments the run is still accepting.

  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 $(s_0, 0)$, ... , (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.

- The case that N is a 2NRW is very similar to 2NBW. There is some index c such that the set G_c is visited infinitely often and the set B_c is visited finitely often.

  Let (s_d, i_d) denote the first visit to G_c after which there are no visits to B_c. For all $j < k < d$ such that $s_j = s_k$ and $i_j = i_k$ we remove the segment of the run between s_j and s_{k-1}. The run clearly stays valid and accepting (finite number of changes). Now just like for Büchi automata, we identify all locations $j < k$ for which $s_j = s_k$ and $i_j = i_k$ and for all $j \leq p \leq k$ we have $s_p \notin G_c$. These run segments are removed from the run. As before, the run obtained in the limit stays valid and accepting.

  Finally, 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 G_c between the two. Clearly B_c cannot occur between s_j and s_k and the segment can be converted into an accepting loop.

  The case that n is a 2NPW is treated similarly. The set F_c takes the role of G_c and $\bigcup_{c'<c} F_c$ takes the role of B_c.

□

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. Given a formula $f \in B^+(S)$, we *dualize* f by replacing $\wedge$ by $\vee$, **true** by **false** and vice versa. The *dual* of f is denoted $\widetilde{f}$.

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.

A *1-way alternating automaton* is $A = \langle \Sigma, Q, s_0, \eta, F \rangle$ where Σ, Q, q_0, and F are like in nondeterministic automata and $\eta : S \times \Sigma \to B^+(Q)$ 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* (*co-Büchi*) *automata* or *1ABW* (*1ACW*) 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 η. Formally, $r(\epsilon) = s_0$ and for all nodes x with $r(x) = s$ and $\eta(s, w_{|x|}) = \varphi$ there is a (possibly empty) set $\{s_1, ..., s_n\} \models \varphi$ such that there are n successors to x, $\{x \cdot 0, \ldots, x \cdot (n-1)\}$ and $x \cdot c$ is labeled by s_{c+1} for $0 \leq c < n$. 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. A run of a 1ACW is *accepting* if every infinite path visits the accepting set finitely 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, $L(A)$.

Given an 1AFA $A = \langle \Sigma, S, s_0, \eta, F \rangle$, the *dual* of A is the 1AFA $\widetilde{A} = \langle \Sigma, S, s_0, \widetilde{\eta}, S \setminus F \rangle$ where $\widetilde{\eta}(s, a)$ is the dual of $\eta(s, a)$. The automata A and $\widetilde{A}$ accept complementary languages [CKS81], i.e. $L(\widetilde{A}) = \Sigma^* \setminus L(A)$. The dualization includes replacing the acceptance condition F by its complement $S \setminus F$. Similarly, given an 1ABW the dualization of the acceptance condition amounts to changing the acceptance mode from Büchi to co-Büchi. Thus, given an 1ABW $A = \langle \Sigma, S, s_0, \eta, F \rangle$, the dual of A is the 1ACW $\widetilde{A} = \langle \Sigma, S, s_0, \widetilde{\eta}, F \rangle$. Again the automata A and $\widetilde{A}$ accept complementary languages [MS87], i.e. $L(\widetilde{A}) = \Sigma^\omega \setminus L(A)$. Note that dualizing an 1AFA results in an 1AFA and dualizing an 1ABW results in an 1ACW. Thus, finding the 1AFA complement of an 1AFA is quite simple, while finding the 1ABW complement of an 1ABW involves a quadratic construction [KV97][2].

We also consider weak alternating automata. A weak alternating automaton (1AWW) is a 1ABW where the set of states S is partitioned into disjoint sets, S_i, such that for each set S_i, either $S_i \subseteq F$, in which case S_i is an *accepting* set, or $S_i \cap F = \emptyset$, in which case S_i is a *rejecting* set. In addition there exists a partial order $\leq$ on the collection of the S_i's such that for every $q \in S_i$ and $s' \in S_j$ for which s' occurs in $\delta(q, a)$, for some $a \in \Sigma$, we have $S_j \leq S_i$. Thus, transitions from a state in S_i lead to states in either the same S_i or a lower one. It follows that every infinite path of a run of a 1AWW ultimately gets "trapped" within some set S_i. The path then satisfies the acceptance condition F if and only if S_i is an accepting set. Thus, we can view 1AWW with acceptance condition F as both a 1ABW with acceptance condition F, and a 1ACW with acceptance condition $S \setminus F$.

[2] Kupferman and Vardi also prove that this construction can not be improved to linear [KV97].

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.

The construction consists of two stages, in the first stage we restrict the automaton so that it can move either forward or backward (and not stay in the same place). In the second stage we convert this 'always moving' automaton into an alternating automaton.

Theorem 3.1 *For every 2NFA $N = \langle \Sigma, S, s_0, \delta, F \rangle$ with n states, there exist 1AFA A and A' with $O(n^2)$ states such that $L(A) = L(N)$ and $L(A') = \Sigma^* \setminus L(N)$.*

Note that if we further convert the 1AFAs into 1NFAs we get automata with $2^{O(n^2)}$ states. The constructions of Vardi and Birget [Var89, Bir93] produce smaller automata.

3.1 Removing ϵ-moves

An *ϵ-move* in a run of a 2NFA 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 an ϵ-move if $i_j = i_{j-1}$.

Our first conversion is from $N = \langle \Sigma, S, s_0, \delta, F \rangle$ with $\delta : S \times \Sigma \to 2^{S \times \{-1,0,1\}}$ to an equivalent $N' = \langle \Sigma, S, s_0, \delta', F \rangle$ with $\delta' : S' \times \Sigma \to 2^{S \times \{-1,1\}}$ such that $L(N) = L(N')$. There are no ϵ-moves in the runs of N'.

We start by defining for each state s and alphabet letter a, the set C_a^s of all states reachable from s by a sequence of ϵ-moves reading letter a and one last forward/backward move.

$$C_a^s = \left\{ (t, \Delta) \in S \times \{-1, 1\} \;\middle|\; \begin{array}{l} \exists k \geq 1 \; \exists (s_0, ..., s_k) \in \{s\} \cdot S^k \\ s.t. \; s_0 = s, \; s_k = t, \\ \forall 0 \leq j < k, \; (s_{j+1}, 0) \in \delta(s_j, a) \\ \text{and } (s_k, \Delta) \in \delta(s_{k-1}, a) \end{array} \right\}$$

Define $\delta'(s, a) = C_a^s$.

Claim 3.2 $L(N) = L(N')$

Proof: Suppose N accepts w. Let $r = (s_0, 0), ..., (s_m, i_m)$ be an accepting run of N on w. We turn r into a run r' of N' on w by pruning ϵ-moves: 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 N' on w.

Suppose N' accepts w. Let $r' = (s_0, 0), ..., (s_m, i_m)$ be an accepting run of N' on w. We append the ϵ-moves from the the appropriate sets C_a^s to complete a run of N on w. $\square$

3.2 Two-way runs

From this point on we consider only 2NFAs with no ϵ-moves. Given a 2NFA $N = \langle \Sigma, S, s_0, \delta, F \rangle$, let $A = \langle \Sigma, Q, s_0, \eta, F \rangle$ denote its equivalent 1AFA. Note that A uses the same acceptance set and initial state as N.

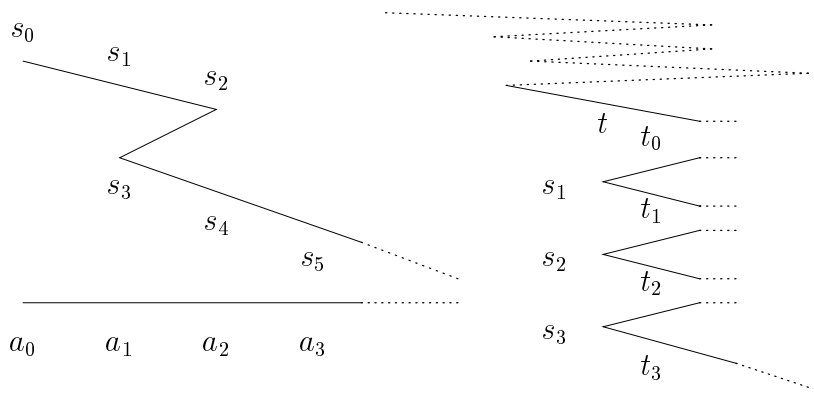

Figure 1: (a) A zigzag run (b) The transition at the singleton state t

Recall that a run of N 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 N our goal is to construct the 1AFA A recognizing the same language. In Figure 1a we see that a run of N 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 *singleton state* $s \in Q$ represents a part of the run that is only looking forward (s_4 in Figure 1a). In fact, we use singleton states to represent only the last forward state in the run of A that visits a letter. 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. Then it is easier to check that $(s_3, -1) \in \delta(s_2, a_2)$.

3.3 The Construction

3.3.1 The transition at a singleton state

We define the transitions of A 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 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 the alternating automaton verifies that a run segment connects t_1 to s_2, etc. In general the alternating 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 read the rest of the word and reach location $|w|$ in an accepting state.

Given a state t and an alphabet letter a, consider the set R_a^t 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

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

The transition of A 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.

$$\eta(t, a) = \bigvee_{\langle t_0, ..., t_k \rangle \in R_a^t} (t_0, s_1) \wedge (t_1, s_2) \wedge ... \wedge (t_{k-1}, s_k) \wedge t_k$$

3.3.2 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.

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 $R_a^{(t,s)}$ 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

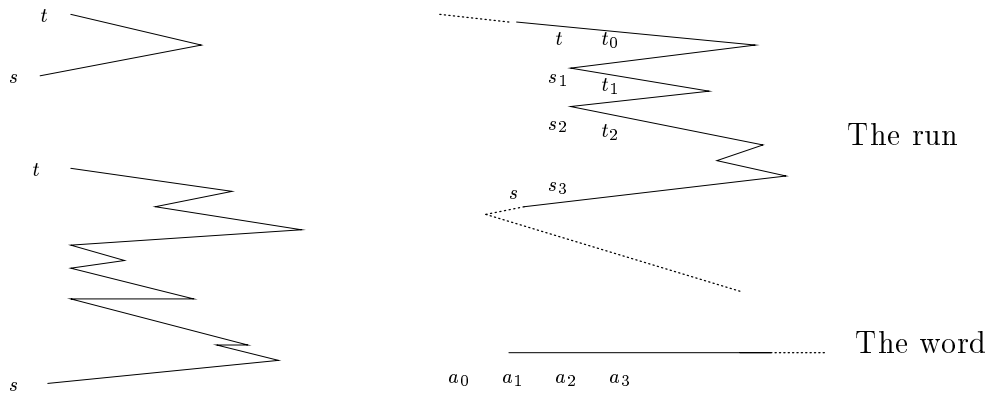

Figure 2: (a) Different connecting segments (b) The transition at the pair state (t, s)

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 ($(t_i, 1) \in \delta(s_i, a)$).

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

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

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

3.4 Proof of correctness

To conclude, the complete description of A is $\langle \Sigma, Q, s_0, \eta, F \rangle$ where the initial state and the set of accepting states is equal to that of N and η is as defined. All the pair-labeled paths in a run of A have to terminate "before falling of the edge of the tape" and the singleton-labeled path must "fall off" with an accepting state.

Claim 3.3 $L(A) = L(N)$

Proof: Given an accepting simple run of N 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 N. 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, r') of A by induction. In addition to the labeling $r' : T \to S \cup (S \times S)$, 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 N. For example the root of the run tree of A is labeled by s_0 and tagged by $(s_0, 0, 0)$. The labeling and the tagging conform to the following:

- Given a node x labeled by state s tagged by (s', i, j) from the run of N we build the tree so that $s = s',\ i = |x|$ and furthermore all triplets in the run of N 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) from the run of N 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 N 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 t, tagged by (t, i, j), and adhering to our demands (see state t in Figure 1b). If (t, i, j) has no successor in the run of N, it must be the case that $i = |w|$ and that $t \in F$. Otherwise we denote the triplets in the run of N 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 N 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 (t, 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 $(t_0, s_1), (t_1, s_2), ..., (t_{k-1}, s_k), t_k$, to a successor of x labeled by (t_{l-1}, s_l) we add the tags $(t_{l-1}, i+1, j_{l-1}+1)$ and (s_l, i, j_l), and to the successor of x labeled by t_k we add the tag $(t_k, i+1, j_k+1)$.

We now show that the new nodes added to the tree conform to our demands. By assumption there are no visits beyond the jth step in the run of N to letters before a_i and $s_1, ..., s_k$ are all the visits to a_i after the jth step of N.

Let y 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 N 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 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 N 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 $(s, i-1, k)$. By assumption there are no visits to a_{i-1} in the run of N 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 N whose third element is between j and k and whose second element is i by $(s_1, i, j_1), ..., (s_m, i, j_m)$ (see Figure 2b). Denote by $(t_1, i+1, j_1+1), ..., (t_m, i+1, j_m+1)$ their successors, by $(t_0, i+1, j+1)$ the successor of t and by $(s_{m+1}, i, k-1)$ the predecessor of s. We add $m+1$ successors to x and label them $(t_0, s_1), (t_1, s_2), ..., (t_m, s_{m+1})$. To a successor of x labeled by (t_{l-1}, s_l) we add the tags $(t_{l-1}, i+1, j_{l-1}+1)$ and (s_l, i, j_l). As in the previous case when we combine the assumption with the way we chose $t_0, ...t_m$ and $s_1, ..., s_{m+1}$, 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 A into a linear run of N. Let (T, r') be an accepting run of A on a word w. We assume ordering on the successors of each node according to the appearance of their labels in the transition of A. The recursive algorithm in Figure 3 constructs an accepting run of N. When first reaching a node x labeled by pair state (t, s), we add t to the run of N. Then we handle recursively the children of x. When we return to x we add s to the run

of N. When reaching a node x labeled by a singleton state s we simply add s to the run of N and handle the sons of x recursively.

```
build_run (x, r'(x) = s, i)
  r := r · ⟨s, i⟩;
  for all sons x · a of x
    build_run (x · a, r'(x · a), i + 1)
  End (for loop)
```

```
build_run (x, r'(x) = (t, s), i)
  r := r · ⟨t, i⟩;
  for all sons x · a of x
    build_run (x · a, r'(x · a), i + 1)
  End (for loop)
  r := r · ⟨s, i − 1⟩;
```

Figure 3: Converting a run of A into a run of N

Starting from the root ϵ labeled $(s_0, 0)$, we add to the run of N 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 N. 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 recursion. When we return to c we add $(s, 0)$ to the run of N (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 $R_{a_0}^{s_0}$ ensures that $(p, 1) \in \delta(s, a_0)$. When we return to ϵ after scanning the whole tree the run of N 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 N. Suppose the last successor of x was labeled (p, q) then from the definition of $R_{a_{|x|}}^{(t,s)}$ 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 A there is a single path labeled solely by singleton states. The last element in the run of N 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 states in the run of N read letters in the range $\{0, ..., |w| - 1\}$. Otherwise there is some node x in the run of A such that $|x| \geq |w|$ (other than the previously designated node). This is impossible since the run of A is accepting. $\square$

As mentioned in Section 2 complementing an 1AFA is simple. The complement of A is $\widetilde{A} = \langle \Sigma, Q, s_0, \widetilde{\eta}, Q \setminus F \rangle$. As $L(A) = L(N)$, $L(\widetilde{A}) = \Sigma^* \setminus L(N)$.

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 N 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 A in the pair-labeled paths. Another problem is when the run ends in a loop.

Theorem 4.1 *For every 2NBW $N = \langle \Sigma, S, s_0, \delta, F \rangle$ with n states, there exist 1ABWs A and A' with $O(n^2)$ states such that $L(A) = L(N)$ and $L(A') = \Sigma^\omega \setminus L(N)$.*

4.1 Removing ϵ-moves

Like in the finite case we first reduce the problem to automata without ϵ-moves. Given an automaton $N = \langle \Sigma, S, s_0, \delta, F \rangle$ where $\delta : S \times \Sigma \to 2^{S \times \{-1,0,1\}}$ we would like to remove all the ϵ-moves. There are two potential problems, visits to F in an ϵ-move and a loop of ϵ-moves that visits F. We double the number of states and add an accepting sink state $N' = \langle \Sigma, (S \times \{\bot, \top\}) \cup \{Acc\}, (s_0, \bot), \delta', (S \times \{\top\}) \cup \{Acc\} \rangle$. A sequence like $..., ((s, \bot), i), ((s', \top), i+1), ...$ in the run means that in the run of N between the appearance of (s, i) and $(s', i+1)$ there was an ϵ-move that visited F. Similarly $\bot$ means that ϵ-moves (if occured) have not visited F (in [Wil99, HK96] similar problems are solved in a similar way).

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 ϵ-moves reading letter a and one last forward/backward step. All states avoid the acceptance set F.

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

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

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

We also have to take care of situations where there is a loop of ϵ-moves 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 $1 \leq k$ and all the following conditions hold.

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

We use the two ϵ-closures and the variable defined above in the definition of the transition function of the 1NFA N'.

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

Apparently, N' is ϵ-move free.

Claim 4.2 *L(N')=L(N)*

Proof: Suppose N accepts w. There exists an accepting run r of N on w. If a finite sequence of ϵ-moves 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 ϵ-moves, 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 N'. It is also an accepting run. If the run ends in a suffix Acc^ω then it is clearly accepting. Otherwise, removing sequences of ϵ-moves 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 N'.

Suppose N' accepts w. We append ϵ-moves 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. □

4.2 The Construction

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 $F' = (S \times \{\top\}) \cup (F \times \{\perp\})$ and the transition function η is defined as follows.

4.2.1 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.

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

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 $\perp$.

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

This is, however, 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 pairs of sequences of at most $2n$ states, where the last state in the two sequences is equal. This repetition closes the loop. In both sequences no two states in an even/odd position are equal. For example, in Figure 4, we see that in state t reading letter a_1, the alternating automaton guesses the sequence $(t_0, s_1), (t_1, s_2)$ and the sequence $(t_2, s_3), (t_3, s_2)$. The last state in both sequences is s_2.

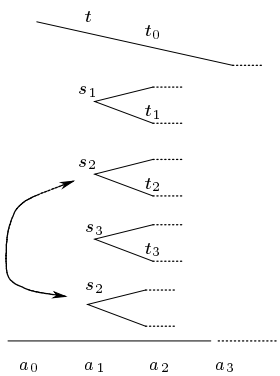

Figure 4: A loop

More formally, we demand that the first state in the first sequence be a successor of t ($(t_0^1, 1) \in \delta(t, a)$), that the first state in the second sequence be a successor of the last state in the first sequence ($(t_0^2, 1) \in \delta(s_{k+1}^1, a)$), that t_i^p be a successor of s_i^p for $p \in \{1, 2\}$ ($(t_i^p, 1) \in \delta(s_i^p, a)$) and that the last state in the first sequence be equal to the last state in the second sequence ($s_{k+1}^1 = s_{l+1}^2$).

$$L_a^t = \left\{ \left\langle \begin{array}{l} \langle t_0^1, s_1^1, t_1^1, ..., s_k^1, t_k^1, s_{k+1}^1 \rangle, \\ \langle t_0^2, s_1^2, t_1^2, ..., s_l^2, t_l^2, s_{l+1}^2 \rangle \end{array} \right\rangle \;\middle|\; \begin{array}{l} 0 \leq k < n,\ 0 \leq l < n \\ (t_0^1, 1) \in \delta(t, a),\ (t_0^2, 1) \in \delta(s_{k+1}^1, a) \\ \forall i < j,\ s_i^1 \neq s_j^1 \text{ and } t_i^1 \neq t_j^1 \\ \forall i < j,\ s_i^2 \neq s_j^2 \text{ and } t_i^2 \neq t_j^2 \\ \forall i,\ \forall p,\ (t_i^p, 1) \in \delta(s_i^p, a) \\ s_{k+1}^1 = s_{l+1}^2 \end{array} \right\}$$

It is obvious that a visit to F has to occur within the loop. Hence we have to make sure that the run segment connecting one of the pairs in the second sequence visits F. Hence we annotate one of the pairs $(t_0^2, s_1^2), ..., (t_l^2, s_{l+1}^2)$ with $\top$. One visit to F is enough hence all other pairs are annotated by $\bot$.

$$\alpha_l^L = \{\langle \alpha_0, ..., \alpha_l \rangle \in \{\bot, \top\}^{l+1} \mid \exists! i \text{ s.t. } \alpha_i = \top\}$$

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

$$\eta((t, \bot), a) = \eta((t, \top), a) = \bigvee \begin{array}{l} \bigvee_{R_a^t, \alpha_k^R} (t_0, s_1, \alpha_0) \wedge ... \wedge (t_{k-1}, s_k, \alpha_{k-1}) \wedge (t_k, \alpha_k) \\ \\ \bigvee_{L_a^t, \alpha_l^L} \left(\begin{array}{l} (t_0^1, s_1^1, \bot) \wedge ... \wedge (t_k^1, s_{k+1}^1, \bot) \wedge \\ (t_0^2, s_1^2, \alpha_0) \wedge ... \wedge (t_l^2, s_{l+1}^2, \alpha_l) \end{array} \right) \end{array}$$

4.2.2 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.

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

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.

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

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

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

$$\eta((t, s, \top), a) = \begin{cases} \textbf{true} & \text{If } (s, -1) \in \delta(t, a) \text{ and } (s \in F \text{ or } t \in F) \\ \bigvee\limits_{R_a^{(t,s)}, \alpha^R_{s,t,k}} (t_0, s_1, \alpha_0) \wedge ... \wedge (t_k, s_{k+1}, \alpha_k) & \text{Otherwise} \end{cases}$$

4.3 Proof of correctness

The proof is just an elaboration on the proof of the finite case. In both directions we use the similar constructions. We only have to give special attention to visits to the accepting set. As the proofs are almost identical we just hilight the points of difference.

Claim 4.3 *L(N)=L(A)*

Proof: Given an accepting simple run of N 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 N. 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 $\perp$ and $\top$.

When dealing with a node x in the run tree of A 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 $\perp$. 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 $\perp$.

When dealing with a node x in the run tree of A 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 $\alpha = \perp$ 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 A. 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 A as before until reaching the node x in level i labeled by a singleton state (s, α) tagged by (s, i, j_0). As letter i is visited in the loop there are infinitely many visits to it. Denote these visits by $(s_0, i, j_0), (s_1, i, j_1), (s_2, i, j_2), ...$, all backward states.

Let s' be the first state in the sequence above that appears infinitely often. Denote by $(s_0^1, i, j_0^1), \ldots, (s_m^1, i, j_m^1)$ the prefix of the sequence until the first occurrence of s'. As the run is simple, there is no other state repeating twice in this prefix. Similarly denote by $(s_0^2, i, j_0^2), \ldots (s_p^2, i, j_p^2)$ the sequence of states from the first occurrence of s' (excluding the occurrence) to the second occurrence of s'. Again, as the run is simple there has to be a visit to F between locations j_0^1 and j_p^2 in the run of N and $p \leq n$. Now denote $t_0^1, \ldots t_{m-1}^1$ the successors of $s_0^1, \ldots, s_{m-1}^1$, t_0^2 the successor of s_m^1, and $t_1^2, \ldots, t_{p-1}^2$ the successors of $s_1^2, \ldots s_{p-1}^2$. We add $m + p$ successors to x and label them $(t_0^1, s_1^1), \ldots (t_{m-1}^1, s_m^1)$ and $(t_0^2, s_1^2), \ldots, (t_{p-1}^2, s_p^2)$. Obviously, all the conditions required in $L_{w_i}^s$ are fulfilled by this pair of sequences. There exists some j_h^2 for which there is a visit to F between locations j_h^2 and j_{h+1}^2 in the run of N. We annotate the pair (t_h^2, s_{h+1}^2) by $\top$ and all other pairs by $\bot$. We tag a pair $(t_d^\alpha, s_{d+1}^\alpha)$ by $(t_d^\alpha, i, j_d^\alpha + 1)$ and $(s_d^\alpha, i, j_{d+1}^\alpha)$.

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

If the accepting run tree of A 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. There are two pair states below x labeled by $(t_1, s, \bot)$ and (t_2, s, β). We start handling the successors of x until we finish handling the successor labeled $(t_1, s, \bot)$. Then, we put aside the run of N built so far and call it r. Now we build a new run r' starting from the point we stopped. Since the run of A is finite the recursion ends and the run r' is finite. As a final step we present $r \cdot (r')^\omega$ as the new run of N. □

Both in the finite and the infinite case we separated the construction into two stages. Namely removing the zero steps and then transforming automata that take no ϵ-moves. In the finite case the first stage did not increase the number of states. In the infinite case the first stage doubled the number of states and then squaring we get approximately $8n^2 + 12n$ states. We could actually unite the two stages of the construction into one stage. Such a construction includes the ϵ-moves in the definition of the sets R_a and L_a. We believe our construction is easier to understand, while improving our construction to include the modification is not so difficult. Transforming the 2NBW into a 1ABW in one stage results in an automaton with approximately $2n^2 + 2n$ states.

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 ϵ-moves in order to reduce the transition to polynomial size [GH96]. Their construction uses the reverse language and can not be applied to infinite words. In Appendices B and C, we use ϵ-moves 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. In the case of unitary alphabet, our construction, with ϵ-moves, gives a polynomial time algorithm for checking the emptiness of 2NBW. For 2NFA a log space algorithm exists [Var90].

4.4 Complementing the alternating automaton

Complementing an 1ABW is not as easy as complementing an 1AFA. In the finite case dualizing the transition function and the acceptance set is enough. In the infinite case we can dualize the transition but instead of Büchi acceptance we have to use co-Büchi acceptance. That is, states from the acceptance set have to appear only finitely often along every infinite path [MS87].

Kupferman and Vardi [KV97] showed how to complement alternating automata using weak alternating automata. Given a 2NBW N with n states, we constructed a 1ABW A with $O(n^2)$ states. If we implement the quadratic construction from [KV97] on A we get A', a 1ABW with $O(n^4)$ states accepting the complementary language of N. We show how to construct an 1ABW with $O(n^2)$ states whose language is the complement of N's language. We recall the proof in [KV97] and show how to avoid the quadratic price in our case. The following observations about runs of 1ACW are taken from [KV97] with minor adjustments.

Definition 4.4 [KV97] *A tree run (T, r) is memoryless if for all $x_1, x_2 \in T$ such that $|x_1| = |x_2|$ and $r(x_1) = r(x_2)$, we have that for all $y \in \mathbb{N}^*$, $r(x_1 \cdot y) = r(x_2 \cdot y)$.*

Theorem 4.5 [EJ91] *If a co-Büchi automaton accepts a word w, then there exists a memoryless accepting run on w.*

We can restrict our attention to memoryless run trees. Hence, the run tree (T, r) can be represented in the form of a directed acyclic graph $G = (V, E)$ where $V \subseteq Q \times \mathbb{N}$ and $E \subseteq \bigcup_{i=0}^{\infty}(Q \times \{i\}) \times (Q \times \{i+1\})$:
$V = \{(V(x), |x|) \mid x \in T\}$
$E = \{((V(x), |x|), (V(y), |y|)) \mid x, y \in T \text{ and } y \text{ successor of } x \text{ in } T\}$

Given a (possibly finite) DAG $G' \subseteq G$. We define a vertex (s, i) as *eventually safe* in G' iff only finitely many vertices in G' are reachable from (s, i). We define a vertex (s, i) as *currently safe* in G' iff all the vertices in G' reachable from (s, i) are not members of $F \times \mathbb{N}$.

Now define the inductive sequence:

- $G_0 = G$
- $G_{2i+1} = G_{2i} \setminus \{(s, i) \mid (s, i) \text{ is eventually safe in } G_{2i}\}$
- $G_{2i+2} = G_{2i+1} \setminus \{(s, i) \mid (s, i) \text{ is currently safe in } G_{2i+1}\}$

Definition 4.6 Border, Ultimate Width

1. *Given a graph G_i and a number $0 \leq p \leq n$ the border of p in G_i is the level $l \in \mathbb{N}$ such that for all $l' \geq l$ there are at most p vertices of the form (s, l) in G_i. If no such number exists then we define the border of p in G_i to be infinity.*

2. *Given a graph G_i the ultimate width of G_i is the minimal number $w \leq n$ such that the border of w in G_i is finite. We denote the ultimate width of G_i by $w(G_i)$.*

Lemma 4.7 [KV97] *For every $i \geq 0$, either $w(G_{2i}) = 0$ or $w(G_{2i+2}) < w(G_{2i})$.*

In our case, we have the 1ABW A. Its complement, the 1ACW $\widetilde{A}$ has the same state set $(S \cup (S \times S)) \times \{\bot, \top\}$. The state set of $\widetilde{A}$ can be partitioned into two sets, $S \times \{\bot, \top\}$ and $S \times S \times \{\bot, \top\}$. The transition of states of the form (s, t, α) includes only states from the same set. This set and the acceptance set do not intersect, hence in the graph G_1 all the states of this form are 'currently safe' and all of them are missing from G_2. We can conclude that $w(G_2) \leq 2|S|$. Therefore, if we denote $2|S|$ by n the graph G_{2n+2} is finite and hence G_{2n+3} is empty.

Index the vertices in G in the following way:

- $2i$, if the vertex is eventually safe in G_{2i}
- $2i+1$ if the vertex is currently safe in G_{2i+1}

All indices are in the range $[2n+2]$.

So we have our co-Büchi automaton $\widetilde{A} = \langle \Sigma, Q, (s_0, \bot), \widetilde{\eta}, F \rangle$ where $Q = (S \cup (S \times S)) \times \{\bot, \top\}$. Kupferman and Vardi show how to construct a weak alternating automaton with state set $Q \times \{0, ..., 2n+2\}$ that accepts the same language (that is the language of $\widetilde{A}$, the complement language of A).

We can further reduce the number of states. Recall that only pair-states are reachable from pair-states and no pair-state is in the acceptance set. Hence we can define G_0 to be $G \setminus (S \times S \times \{\bot, \top\} \times \mathbb{N})$ i.e. remove from G all the pair labeled states (which are currently safe in G). This way all indices are in the range $[2n]$. Furthermore there is no need to multiply all the states in Q by $[2n]$. It is enough to multiply $S \times \{\bot, \top\}$ by $[2n]$ and consider $(S \times S \times \{\bot, \top\}$ as the minimal set of the weak alternating automaton.

To conclude we give the final weak alternating automaton accepting the language of $\widetilde{A}$ (the complement of A). Given $A = \langle \Sigma, Q, (s_0, \bot), \eta, F \rangle$ where $Q = (S \cup (S \times S)) \times \{\bot, \top\}$ we define $\overline{A} = \langle \Sigma, Q', s_0', \overline{\eta}, F' \rangle$ where $Q' = (S \times \{\bot, \top\} \times [2n]) \cup (S \times S \times \{\bot, \top\})$ where $n = 2|S|$. We follow the notation from [KV97] and define $release : B^+(Q) \times [2n] \to B^+(Q')$. Given a formula $\varphi \in B^+(Q)$, and a rank $i \in [2n]$, the formula $release(\varphi, i)$ is obtained from φ by replacing every atom of the form (s, α) from $S \times \{\bot, \top\}$ by $\bigvee_{l \leq i}(s, \alpha, l)$. Let $\widetilde{\eta}$ be the dualization of η then:

$$\overline{\eta}((s, \alpha, i), a) = \begin{cases} release(\widetilde{\eta}((s, \alpha), a), i) & \text{if } (s, \alpha) \notin F \text{ or } i \text{ is even} \\ \textbf{false} & \text{if } (s, \alpha) \in F \text{ and } i \text{ is odd} \end{cases}$$

$$\overline{\eta}((s, t, \alpha), a) = \widetilde{\eta}((s, t, \alpha), a)$$

Finally $s_0' = (s_0, \bot, 2n)$ and $F' = \{(s, \alpha, i) | i \text{ is odd}\} \cup (S \times S \times \{\bot, \top\})$.

4.5 Parity and Rabin acceptance conditions

Our method works also for 2-way nondeterministic Rabin automata and 2-way nondeterministic Parity automata.

Theorem 4.8 *For every 2-way nondeterministic Rabin (parity) automaton $N = \langle \Sigma, S, s_0, \delta, \alpha \rangle$ with n states and index m, there exists a 1ABW A with $O(n^2 \cdot m)$ states such that $L(A) = L(N)$.*

Given a 2NRW $N = \langle \Sigma, S, s_0, \delta, \alpha \rangle$ where $\alpha = \{\langle G_1, B_1 \rangle, ..., \langle G_m, B_m \rangle\}$ with n states it is straightforward to construct an equal 2NBW N' with $O(n \cdot m)$ states. The construction is not different from the conversion of 1-way nondeterministic Rabin automata to 1-way nondeterministic Büchi automata [Cho74]. Converting the 2NBW N' to a 1ABW A, results in a 1ABW with $O(n^2 \cdot m^2)$ states.

This construction can be improved as follows. Build a 1ABW A for N (without constructing N' first). Multiply the state set of A by the index (and one extra copy) $m+1$. The ith copy of the automaton avoids all the states in B_i. The alternating automaton starts running in copy 0. The transition at a singleton state in copy 0 includes also a guess whether to stay in copy 0 or guess that states from B_i are not visited again during the run and then move to copy i. We should allow also moving into copy i in the second sequence in the transition of a loop. In this case only the part of the loop itself should avoid B_i and should include a demand for visiting G_i. The transition at a state from the ith copy includes only states of the same copy. Reference to the accepting set should be made only outside of copy 0 and in this case G_i serves as F.

For 2NPW the changes to the construction are very similar to the ones described above.

5 Conclusions

We have shown two constructions. Both show how to construct a 1-way alternating automaton that accepts the same language as a 2-way nondeterministic automaton. The first construction for automata that work on finite words and the second for automata that work on infinite words.

In the finite case complementation of alternating automata is very easy. Hence we can easily get the automaton recognizing the complementary language. This automaton can be envisioned as searching for errors in all the possible zigzagging run.

The number of states of the 1AFA is quadratic in the number of states of the 2NFA and the size of the transition is exponential in the size of the original transition. If we further convert our 1AFA into a nondeterministic automaton we get an automaton with $2^{O(n^2)}$ states. Birget and Vardi [Bir93, Var89] showed that given a 2NFA, it is possible to construct 1NFA recognizing the same language and the complementary language with $2^{O(n)}$ states. Given a 2NFA automaton and seeking a 1NFA one should obviously choose their constructions.

In the infinite case we get similar results. Given a 2NBW with n states we get an 1ABW with $O(n^2)$ states. If we use the construction in [MH84], we get a 1NBW with $2^{O(n^2)}$ states. As mentioned Vardi has already solved this problem [Var88]. He shows, given a 2NBW, how to construct two 1NBW, one accepting the same language and one the complementary language, both with $2^{O(n^2)}$ states.

6 Acknowledgments

We would like to thank Orna Kupferman for her remarks on the manuscript and an anonymous referee for pointing out the works of Ruzzo and Venkateswaran.

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.

[Cho74] Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and System Sciences*, 8:117–141, 1974.

[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.

[EJ91] E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 368–377, San Juan, October 1991.

[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 Verification 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 Verification of Coordinating Processes.* Princeton Univ. Press, 1994.

[KV97] O. Kupferman and M.Y. Vardi. Weak alternating automata are not that weak. In *Proc. 5th Israeli Symp. on Theory of Computing and Systems*, pages 147–158. IEEE Computer Society Press, 1997.

[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] R.E. Ladner, R.J. Lipton, and L.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.

[MS87] D.E. Muller and P.E. Schupp. Alternating automata on infinite trees. *Theoretical Computer Science*, 54:267–276, 1987.

[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.

[Ruz80] W. Ruzzo. Tree-size bounded alternation. *Journal of Computer and System Sciences*, 21:218–235, 1980.

[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 *10th ACM Symposium on Theory of Computing*, pages 275–286, San Diego, California, May 1978.

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

[Var90] M.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.

[Ven91] H. Venkateswaran. Properties that characterize LOGCFL. *Journal of Computer and System Sciences*, 43(2):380–404, 1991.

[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 Alternating Automata with ϵ-moves

An alternating automaton with ϵ-moves $A = \langle \Sigma, S, s_0, \eta, F \rangle$ is very similar to the alternating automata defined in Section 2. The only difference is that it can pose demands also on the same location in the word and not only on the next location. Formally, the transition function $\eta : S \times \Sigma \to B^+(\{0,1\} \times S)$ has every state labeled by the direction 0 or 1. The run of such an automaton is still a labeled tree (T, r). The depth in the tree no longer corresponds to the location in the word. Thus,the labeling r associates with every node $x \in T$ a pair $(s, i) \in S \times \mathbb{N}$ where s is the state and i is the location in w.

A run of such an automaton on a finite word $w = w_0, \ldots, w_{m-1}$ is accepting if all the labels are in $S \times \{0 \ldots m\}$ and all the nodes labeled by location m are accepting states. A run of such an automaton on an infinite word is accepting if all infinite paths visit $F \times \mathbb{N}$ infinitely often.

B 2NFA to 1AFA with ϵ-moves

As one may expect, the construction with ϵ-moves is very similar to the previous construction. Instead of guessing in one step all the visits to the next letter, we guess whether there exists another visit to this letter. In such a case, the automaton spawns two states, a singleton state that is responsible for the rest of the run and a pair-state that is responsible for the connection between the current state and the next visit to the same letter. The run segment connecting the two may not visit letters before the current letter.

Spawning states that can read the same letter has two advantages. We do not have to use the notion of forward states and backward states. A state reading letter i in the run of the 2NFA reads letter i in the run of the 1AFA (unlike before where we have backward states reading letter $j-1$ in the run of the 2NFA associated with letter j in the run of the 1AFA). We can also treat ϵ-moves of the 2NFA very easily, by having ϵ-moves of the 1AFA.

On the other hand, we have a problem checking backward moves. When the 1AFA follows a backward move it does not know the letter the move depends on. In order to solve this problem we introduce states of the form $s \to t$ for s and t states of the 2NFA. Such a state means that we can get from state s to state t by a sequence of ϵ-moves followed by one backward move[3].

Given a 2NFA $N = \langle \Sigma, S, s_0, \delta, F \rangle$ we construct an 1AFA with ϵ-moves $A = \langle \Sigma, Q, s_0, \eta, F \rangle$ such that $L(A) = L(N)$. Our 1AFA uses the initial state and the acceptance set of the 2NFA. The set of states is $Q = (S \cup (S \times S)) \cup \{s \to t \mid s, t \in S\}$, and the transition function η is defined for every state in Q and letter in Σ as follows.

$$\eta(t,a) = \bigvee \begin{cases} \bigvee_{(s,0) \in \delta(t,a)} (s,0) \\ \\ \bigvee_{s \in S} ((t,s),0) \wedge (s,0) \\ \\ \bigvee_{(s,1) \in \delta(t,a)} (s,1) \end{cases}$$

[3]Notice that if $|\Sigma| < |S|^2$ it makes more sense to guess the next letter, check that using the guessed letter we can get from s to t using ϵ-moves and one backward move. Finally, make sure that the next letter is indeed equal to the guessed letter. In particular for 1-letter alphabet, there is no need for adding extra states.

$$\eta((t,s),a) = \bigvee \begin{cases} \bigvee_{(t_1,0)\in\delta(t,a)} ((t_1,s),0) \\ \\ \bigvee_{s_1\in S} ((t,s_1),0) \wedge ((s_1,s),0) \\ \\ \bigvee_{(t_1,1)\in\delta(t,a)} \bigvee_{s_1\in S} ((t_1,s_1),1) \wedge (s_1 \to s,1) \end{cases}$$

$$\eta(s_1 \to s,a) = \begin{cases} \textbf{true} & (s,-1) \in \delta(s_1,a) \\ \\ \bigvee_{(s_2,0)\in\delta(s_1,a)} (s_2 \to s,0) \end{cases}$$

In state t reading letter i the 1AFA can (a) move using an ϵ-move of N, (b) guess that there is some other visit to letter i in state s and spawn two states (t,s) and s both reading letter i, or (c) guess that there is no other visit to letter i and use a forward transition of N.

In state (t,s) reading letter i the 1AFA can (a) move from t using an ϵ-move of N, (b) guess that there is some visit to letter i between t and s in state s_1 and spawn two states (t,s_1) and (s_1,s) both reading letter i, or (c) guess that there is no other visit to letter i between t and s and use a forward transition of N from state t and guess that there is a backward transition moving to state s. From state $s_1 \to s$ the automaton either takes an ϵ-move from s_1 or a backward step from s_1 to s, using the next letter. Finally, we replace every occurrence of (t,t) in η by $true$.

The proof that $L(A) = L(N)$ is very similar to the previous proof. Notice that a state appearing in the run of N only once, may appear many times in the run of A. When converting a run of A into a run of N such states should be added only once.

Finally, denote $|S| = n$ and $|\delta| = m$. We have, $|Q| = O(n^2)$ and $|\eta| = O(m \cdot n^2)$.

C 2NBW to 2ABW with ϵ-moves

We enhance the construction in Appendix B to work for 2NBW. Again we annotate each state by $\bot$ and $\top$. A singleton state annotated by $\top$ signifies a visit to the acceptance set in the run segment connecting it to the previous singleton state. A pair-state annotated by $\top$ is a promise to visit the acceptance set in the run segment connecting the two states.

In the set $\{\bot,\top\}$ consider $\bot + \bot = \bot$, $\bot + \top = \top + \bot = \top$, and $\top + \top$ as undefined.

Given a 2NBW $N = \langle \Sigma, S, s_0, \delta, F\rangle$ we construct an 1ABW with ϵ-moves $A = \langle \Sigma, Q, q_0, \eta, F'\rangle$ such that $L(A) = L(N)$. Where $Q = ((S \cup (S \times S)) \times \{\bot,\top\}) \cup \{s \to t \mid s,t \in S\}$, $q_0 = (s_0,\bot)$, $F' = F \times \{\bot\} \cup S \times \{\top\}$ and η is defined for every state in Q and letter in Σ as follows. First we define two functions $f_\alpha : S \times S \to \{\bot,\top\}$ where $\alpha \in \{\bot,\top\}$.

$$f_\bot(s,t) = \bot$$

$$f_\top(s,t) = \begin{cases} \bot & s \in F \text{ or } t \in F \\ \top & \text{Otherwise} \end{cases}$$

$$\eta((t,\alpha),a) = \bigvee \begin{cases} \bigvee_{(s,0)\in\delta(t,a)} ((s,\bot),0) \\ \\ \bigvee_{s\in S} \bigvee_{\beta\in\{\bot,\top\}} ((t,s,f_\beta(t,s)),0) \wedge ((s,\beta),0) \\ \\ \bigvee_{(s,1)\in\delta(t,a)} ((s,\bot),1) \end{cases}$$

$$\eta((t,s,\alpha),a) = \bigvee \begin{cases} \bigvee_{(t_1,0)\in\delta(t,a)} ((t_1,s,f_\alpha(t_1,s)),0) \\ \\ \bigvee_{s_1\in S} \bigvee_{\beta_1+\beta_2=\alpha} ((t,s_1,f_{\beta_1}(t,s_1)),0) \wedge ((s_1,s,f_{\beta_2}(s_1,s)),0) \\ \\ \bigvee_{(t_1,1)\in\delta(t,a)} \bigvee_{s_1\in S} ((t_1,s_1,f_\alpha(t_1,s_1)),1) \wedge (s_1 \to s,1) \end{cases}$$

$$\eta(s_1 \to s,a) = \begin{cases} true & (s,-1)\in\delta(s_1,a) \\ \\ \bigvee_{(s_2,0)\in\delta(s_1,a)} (s_2 \to s,0) \end{cases}$$

Finally, we replace in η every occurrence of $(t,t,\top)$ and $(f,f,\bot)$ where $f \in F$ by **true**.

Again the proof that $L(A) = L(N)$ is not very different from previous proofs. If we denote $|S| = n$ and $|\delta| = m$, we have, $|Q| = O(n^2)$ and $|\eta| = O(m \cdot n^2)$.

The construction of the weak automaton that complements A is not modified by the presence of ϵ-moves. Formally, $\overline{A} = \langle \Sigma, Q', q_0', \overline{\eta}, F' \rangle$ where

$$Q' = (S \times S \times \{\bot,\top\}) \cup (S \times \{\bot,\top\} \times [2n]) \cup \{s \to t\}$$

$$q_0 = (s_0, \bot, 2n)$$

$$F' = (S \times \{\bot,\top\} \times [2n]^{odd}) \cup (S \times S \times \{\bot,\top\}) \cup \{s \to t\}$$

$$\overline{\eta}((t,\alpha,i),a) = release(\widetilde{\eta}((t,\alpha),a),i)$$

$$\overline{\eta}((t,s,\alpha),a) = \widetilde{\eta}((t,s,\alpha),a)$$

The partition includes $S \times S \times \{\bot,\top\}$ and $s \to t$ as the minimal sets.

The size analysis does not change and we still have $|Q'| = O(n^2)$ and $|\overline{\eta}| = O(m \cdot n^2)$.