Extending temporal logic with ω-automata

Thesis for the M.Sc. Degree

by

Nir Piterman

Under the Supervision of
Prof. Amir Pnueli
Department of Computer Science
The Weizmann Institute of Science
Prof. Moshe Vardi
Department of Computer Science
Rice University

Submitted to the Feinberg Graduate School of
the Weizmann Institute of Science
Rehovot 76100, Israel

August 22, 2000

Abstract

We investigate the extension of linear temporal logic with ω-automata. We give an alternative translation from Extended Temporal Logic [WVS83] formulas to nondeterministic Büchi automata. The novelty in our translation is usage of alternating automata, thus, simplifying the translation while staying with the same complexity bounds.

We continue and use alternating Büchi automata as temporal connectives of the logic. Again we translate the formulas of the logic to nondeterministic Büchi automata. Although alternating automata are exponentialy more succinct than nondeterministic ones, the complexity of the translation does not change.

Finally we combine the extension in the expressive power of the logic with the reference to the past. We use 2-way alternating automata as temporal connectives. Also here we give a translation of logic formulas to nondeterministic Büchi automata.

Contents

Chapter 1

Preliminaries

1.1 Introduction

Temporal logic has been used for many years now as a tool for the specification and verification of programs [Pnu77, MP92]. Although as expressive as monadic first-order logic of the natural numbers with the less than relation, Wolper [Wol83] has shown that for the task of verification temporal logic is sometimes not expressive enough.

Wolper [Wol83] suggested to augment temporal logic with the power of the ω-regular expressions. Wolper, Vardi and Sistla followed and considered ω-automata as a finitary way of representing the ω-regular expressions [WVS83, SVW87, VW94]. They created several logics, using different types of automata. Safra and Vardi tried to find which automata produce the most succinct formulas [SV89].

Extending temporal logic with ω-automata seems like a reasonable proposition. Hardware implementations frequently include Finite State Machines (FSMs). As automata and FSMs are basically the same thing, it seems that including FSMs in the specification language will give the implementers a powerful formalism they are already familiar with.

ω-automata are characterized by different acceptance conditions. Wolper et al. [WVS83, SVW87, VW94] proposed to use nondeterministic finite automata (yielding the logic ETL_f), nondeterministic looping automata (ETL_l) or nondeterministic Büchi automata (ETL_r). We call these logics the extended temporal logics or ETLs for short. Wolper et al. studied the expressive powers of the different ETLs. They showed that every ω-regular set can be expressed by a formula of every one of the ETLs and that the set of models of an ETL formula is an ω-regular set [WVS83, VW94].

Given a logic formula, an important question is whether it is satisfiable. This question was studied for the three mentioned extended temporal logics. Decision procedures for the logics were offered. A formula of the logic was converted into a nondeterministic Büchi automaton such that the automaton accepts exactly the set of models of the formula [Var96]. Hence, the formula is satisfiable iff the automaton's language is not empty. The decision problem for each of the logics is shown to be PSPACE-complete. The decision problem for ETL_f and ETL_l is in linear nondeterministic space [WVS83, VW94]. The decision problem for ETL_r is in nondeterministic $O(n^2)$ space [SVW87].

Our main interest in this work is the decision problem for temporal logic extended with ω-automata. Since the publication of the above mentioned papers, alternating automata [CKS81, BL80] were introduced and widely studied. Since the combinatorial structure of alternating automata is rich, they are more suitable for handling logic than nondeterministic automata. Alter-

nating automata enable a complete partition between the logical and the combinatorial aspects of the decision problem for logic, and give rise to cleaner and simpler algorithms [Var96].

The decision procedures for ETL_f, ETL_l, and ETL_r used an ad-hoc construction of a nondeterministic Büchi automaton. We propose a more uniform treatment. Given a formula we first translate it into an alternating Büchi automaton with the same set of models. This alternating automaton can be converted to a nondeterministic Büchi automaton using the construction of Miyano and Hayashi [MH84]. The usage of alternating automata yields a cleaner construction with cleaner proofs. We stay within the same complexity bounds and improve the decision procedure of ETL_r to $O(n\ log(n))$ nondeterministic space (using complementation constructions for nondeterministic Büchi automata [KV97, Tho98, Saf88]).

Safra and Vardi [SV89] checked other types of automata, they tried to find the most succinct way of writing formulas. As suggested in [VW94], we use alternating automata as temporal connectives. It was shown that nondeterministic automata and alternating automata have the same expressive power [MH84], hence temporal logic extended with nondeterministic automata is just as expressive as temporal logic extended with alternating automata. On the other hand, alternating automata are exponentially more succinct than nondeterministic automata. There are languages that can be recognized by an alternating automaton with n states but nondeterministic automata recognizing these languages have at least $exp(n)$ states [BL80, CKS81].

We solve the decision problem of this logic in the same way. We translate a formula to an alternating automaton whose language is exactly the set of models of the formula. This alternating automaton in turn is translated to a nondeterministic automaton that can be checked for emptiness.

Our final problem with ETL is that it cannot express properties that depend on the past. It was shown that temporal logic with past operators is more adequate to the task of compositional verification [LPZ85]. We can solve the expressiveness problem and add reference to past properties by introducing 2-way alternating automata as temporal connectives. Vardi [Var98] has shown how to transform a 2-way alternating automaton to a 1-way nondeterministic automaton. We slightly modify his work and get a transformation from 2-way alternating automata to 1-way alternating automata. We incorporate this transformation into the decision procedure for temporal logic augmented with 2-way alternating automata.

1.2 Related Work

Wolper [Wol83] has shown that temporal logic with until and next-time operators cannot express the property "p is true in all even positions", for a proposition p. The ability to count modulo n, not possessed by temporal logic, is important to program specification. Consider the following example of two processes working synchronously that use a single critical section, based on [LPZ85].

$$\left[\begin{array}{ll} l_0: & \text{loop forever} \\ & l_1:\ \text{send ; non critical} \\ & l_2:\ \text{send ; critical} \end{array}\right] \Big\| \left[\begin{array}{ll} m_0: & \text{loop forever} \\ & m_1:\ \text{receive ; critical} \\ & m_2:\ \text{receive ; non critical} \end{array}\right]$$

Before executing 'send' process 1 waits for process 2 to get to 'receive' and vice versa. We would like to establish that process 1 and process 2 never enter the critical section together. The solution proposed in [LPZ85] is to show that process 1 may visit l_2 only after an even number of communications and process 2 may visit m_1 only after an odd number of communications.

Various solutions have been considered for extending the power of temporal logic. For example, using quantifiers ranging over propositional variables [LPZ85, SVW87, MP92], or adding least fixed point operators, resulting in μ-calculus [Koz83].

Wolper [Wol83] suggested extending the expressive power of temporal logic using ω-regular expressions as following. Given a sequence of propositional formulas $f = f_0, f_1, ...$ and a computation $w = w_0, w_1, ...$, we say that the sequence f is satisfied by w if f_0 is satisfied by w_0, f_1 is satisfied by w_1 and so on. Now consider an ω-regular expression over propositional formulas S. The expression S identifies a set of sequences of formulas. We say that S is satisfied by the computation w if there is a sequence of formulas f in S that is satisfied by w.

ω-regular sets can be represented using ω-automata. Wolper's work was extended in [WVS83, VW94], where different automata are suggested as connectives. Wolper et al. study three types of automata. Looping automata (the automaton has to run forever) inducing the logic ETL_l, finite automata (the automaton has to reach a designated set of states) inducing ETL_f and repeating (or Büchi) automata (the set of designated states has to be visited infinitely often) inducing ETL_r.

Wolper et al. [WVS83, VW94] show that the three logics have the same expressive power. Translations from ETL_r to ETL_f and ETL_l are exponential in the size of the formula. The decision problem of formulas is reduced to the emptiness problem of nondeterministic Büchi automata. The nondeterministic automata created are exponential in the size of the formula, yielding a PSPACE algorithm for the decision problem. It should be noted that complementation of Büchi automata results in an exponential blowup that is provably with a nonlinear exponent [Mic88]. Formulas of ETL_r may include negations in front of automata connectives, it seems reasonable that the decision procedure for ETL_r will be in nonlinear space.

Safra and Vardi [SV89] further studied this type of extensions. They extended the logic with Streett automata [Str82] and with EL automata [EL87]. They show that the decision procedure for ETL_{EL} is EXPSPACE-complete. The decision of ETL_S remains in PSPACE and is proposed as the ultimate extended temporal logic.

Another way to classify ω-automata is by the type of their branching mode. In a deterministic automaton, the transition function δ maps a pair of a state and a letter into a single state. The intuition is that when the automaton is in state q and it reads a letter a , then the automaton moves to state $\delta(q, a)$ from which it should accept the suffix of the word. When the branching mode is existential or universal, δ maps q and a into a set of states. In the existential mode, the automaton should accept the suffix of the word from one of the states of the set, and in the universal mode, it should accept the suffix from all the states in the set. In an alternating automaton [CKS81, BL80], both universal and existential modes are allowed, and the transitions are given as Boolean formulas over the set of states. For example $\delta(q, a) = (q_1 \wedge q_2) \vee q_3$ means that the automaton should accept the suffix of the word either from both q_1 and q_2 or from q_3.

Although alternating Büchi automata have the same expressive power as nondeterministic Büchi automata [MH84], they are exponentially more succinct. As suggested in [VW94], we augment temporal logic with alternating automata connectives. Alternating Büchi automata are as expressive as nondeterministic Büchi automata so extending the logic with alternating automata does not change its expressive power. We show that it also does not change the complexity of the decision procedure. We show that a formula of temporal logic extended with alternating automata can be translated to a nondeterministic Büchi automaton with the same complexity as an ETL_r formula.

Two-way automata over infinite structures were introduced as part of the effort to create automata-theoretic techniques to handle μ-calculus with both forward and backward modalities.

Two-way automata over finite words have been shown to have the same power as 1-way automata over finite words [RS59, She59]. Vardi [Var88, Var98] has shown that the same is also true for 2-way automata over infinite structures. Thus, extending temporal logic with 2-way alternating automata results in a logic with the same expressive power. The complexity of this logic is slightly higher and we show that it is decidable in $O(n^2 log(n))$ nondeterministic space.

1.3 Basic Definitions

We consider infinite sequences of symbols from some finite alphabet Σ. Given a *word* w, an element in Σ^ω, we denote by w_i the i^{th} letter of the word w, and by $w_{\geq i}$ the suffix of w starting at w_i hence $w = w_{\geq 0} = w_0 w_1 w_2 \ldots$ and $lim(w) = \{a \in \Sigma | a = w_i \text{ for infinitely many i's}\}$, thus $lim(w)$ is the set of letters appearing infinitely often in w. Automata that read infinite sequnces are usually referred to as ω-automata. We give definitions of three different types of automata.

1.3.1 Finite automata on infinite words

Nondeterministic automata A *nondeterministic automaton* is a five-tuple $A = \langle \Sigma, S, S_0, \rho, F \rangle$, where Σ is the finite alphabet, S is the finite set of states, $S_0 \subseteq S$ is the set of initial states, $\rho : S \times \Sigma \to 2^S$ is the transition function, and F is the acceptance set. We define a *run* of an automaton on an infinite word $w = w_0 w_1 ... \in \Sigma^\omega$ as a finite or infinite sequence $\sigma = s_0, s_1, ...,$ where $s_0 \in S_0$ and for all $0 \leq i < |\sigma|$, we have $s_{i+1} \in \rho(s_i, w_i)$. *Acceptance* of a run is defined according to one of the following conditions:

- *Finite* acceptance, where a state of the set F has to occur somewhere along the run (in this case the run is finite).
- *Looping* acceptance, where the run should be infinite.
- *Repeating* acceptance, where a state of the set F has to occur infinitely often in the run (also called *Büchi* condition).

Alternating automata 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.

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, the nodes $x \cdot y$ where $y \in \mathbb{N}^*$ are the *descendants* 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. We restrict our attention to *finitely branching* trees, forall $x \in T$ the number of successors of x is finite.

An *alternating Büchi automaton* is a five-tuple $A = \langle \Sigma, S, s_0, \rho, F \rangle$ where Σ, S and F are like before. The state s_0 is a unique starting state and $\rho : S \times \Sigma \to B^+(S)$ is the transition function. We define a *run* of an alternating automaton on an infinite word $w = w_0 w_1 ... \in \Sigma^\omega$ as a S-labeled tree (T, V) , where $V(\epsilon) = s_0$ and for all $x \in T$ the (possibly empty) set $\{V(x \cdot c) | c \in \mathbb{N}$ and $x \cdot c \in T\}$ satisfies the formula $\rho(V(x), w_{|x|})$. A run is *accepting* if every infinite path visits the accepting set infinitely often.

A *co-Büchi alternating automaton* is defined exactly the same except that a run is accepting if all infinite paths visit F finitely often.

Given an alternating Büchi automaton $A = \langle \Sigma, S, s_0, \rho, F \rangle$, the dual of A is the co-Büchi automaton $A^d = \langle \Sigma, S, s_0, \rho^d, F \rangle$ where $\rho^d(s, a)$ is the dual of $\rho(s, a)$. The automata A and A^d accpet complementary languages [MS87], i.e. $L(A^d) = \Sigma^\omega \setminus L(A)$.

Two-way alternating automata on infinite words A *2-way alternating Büchi automaton* on infinite words is a five-tuple $A = \langle \Sigma, S, s_0, \rho, F \rangle$ where Σ, S, s_0 and F are like before. The transition function is $\rho : S \times \Sigma \to B^+(\{-1, 0, 1\} \times S)$. A *run* of an automaton on an infinite word $w = w_0 w_1 ... \in \Sigma^\omega$ is a S-labeled tree (T, V), where $V(\epsilon) = (s_0, 0)$ and for all $x \in T$ with $V(x) = (r, n_2)$, the set $\{(s, a) | c \in \mathbb{N}$, $x \cdot c \in T$, $V(x \cdot c) = (s, n_1)$, $a = n_1 - n_2\}$ satisfies the formula $\rho(r, w_{n_2})$. A run is *accepting* if all infinite paths visit F infinitely often.

A *2-way alternating co-Büchi automaton* is defined exactly the same except that a run is accepting if all infinite paths visit F finitely often.

As before, given a 2-way alternating Büchi automaton A, its dual automaton A^d, defined just like for 1-way alternating automata, recognizes the complementary language.

1.3.2 Linear Temporal Logic

We give a short introduction to linear temporal logic (LTL) [Pnu77]. We only mention a formula of this logic once in this paper, as an example. Nevertheless, all this paper is based on this definition.

Syntax Formulas are defined with respect to a set $Prop$ of propositions.

- Every proposition $p \in Prop$ is a formula.
- If f_1 and f_2 are formulas, then $\neg f_1$, $f_1 \vee f_2$, $f_1 \wedge f_2$, $\bigcirc f_1$ and $f_1 U f_2$ are formulas.

Semantics The *satisfaction* of a formula is defined with respect to a model $\pi \in (2^{\mathcal{PROP}})^\omega$ and a location $i \in \mathbb{N}$. We use $(\pi, i) \models \psi$ to indicate that the word π in the designated location i satisfies the formula ψ.

- For a proposition $p \in \mathcal{PROP}$, we have $(\pi, i) \models p$ iff $p \in \pi_i$.
- $(\pi, i) \models \neg f_1$ iff not $(\pi, i) \models f_1$.
- $(\pi, i) \models f_1 \vee f_2$ iff $(\pi, i) \models f_1$ or $(\pi, i) \models f_2$.
- $(\pi, i) \models f_1 \wedge f_2$ iff $(\pi, i) \models f_1$ and $(\pi, i) \models f_2$.
- $(\pi, i) \models \bigcirc f_1$ iff $(\pi, i + 1) \models f_1$.

- $(\pi, i) \models f_1 U f_2$ iff there exists $k \geq i$ such that $(\pi, k) \models f_2$ and for all $i \leq j < k$, we have $(\pi, j) \models f_1$.

We also use the common notations $\Diamond f \equiv trueUf$, for *eventually* f, and $\Box f \equiv \neg\Diamond\neg f$, for *always* f.

1.3.3 Extended Temporal Logic

We present the logics ETL_f, ETL_l and ETL_r as defined in [VW94].

Syntax Formulas are defined with respect to a set $Prop$ of propositions.

- Every proposition $p \in Prop$ is a formula.
- If f_1 and f_2 are formulas, then $\neg f_1$, $f_1 \vee f_2$ and $f_1 \wedge f_2$ are formulas.
- For every nondeterministic finite automaton $A = \langle \Sigma, S, \rho, S_0, F \rangle$ with $\Sigma = \{a_1, ..., a_n\}$. If $f_1, ..., f_n$ are formulas, then $A(f_1, ..., f_n)$ is a formula.

Semantics The *satisfaction* of a formula is defined with respect to a model $\pi \in (2^{\mathcal{PROP}})^\omega$ and a location $i \in \mathbb{N}$. We use $(\pi, i) \models \psi$ to indicate that the word π in the designated location i satisfies the formula ψ.

- For a proposition $p \in \mathcal{PROP}$, we have $(\pi, i) \models p$ iff $p \in \pi_i$.
- $(\pi, i) \models \neg f_1$ iff not $(\pi, i) \models f_1$.
- $(\pi, i) \models f_1 \vee f_2$ iff $(\pi, i) \models f_1$ or $(\pi, i) \models f_2$.
- $(\pi, i) \models f_1 \wedge f_2$ iff $(\pi, i) \models f_1$ and $(\pi, i) \models f_2$.

Consider an automaton $A = \langle \Sigma, S, S_0, \rho, F \rangle$. The *run* of the formula $A(f_1, ..., f_n)$ over a word π starting at point i, is a finite or infinite sequence $\sigma = s_0, s_1, ...$ of states from S, such that $s_0 \in S_0$ and for all k, $0 \leq k \leq |\sigma|$, there is some $a_j \in \Sigma$ such that $(\pi, i + k) \models f_j$ and $s_{k+1} \in \rho(s_k, a_j)$.

We can now complete the definition of semantics:

- $(\pi, i) \models A(f_1, ..., f_n)$ iff there is an accepting run of $A(f_1, ..., f_n)$ over π starting at i.

Yet to be defined is the type of the acceptance used by the automaton: finite, looping, and repeating acceptance induce the logics ETL_f, ETL_l, and ETL_r, respectively.

For example consider the automaton $A = \langle \Sigma, S, S_0, \rho, F \rangle$, where $\Sigma = \{a, b\}$, $S = \{s_0, s_1\}$, $\rho(s_0, a) = \rho(s_1, a) = \{s_0\}$, $\rho(s_0, b) = \rho(s_1, b) = \{s_1\}$, and $S_0 = F = \{s_1\}$. If we consider repeating acceptance, a run of the automaton is accepting if it visits state s_1 infinitely often. The automaton visits s_1 exactly when it reads the letter b. Hence, the ETL_r connective $A(\neg f, f)$ is true iff f is true infinitely often. That is, the ETL_r formula $A(\neg f, f)$ is equal to the LTL formula $\Box\Diamond f$. Other examples can be found in [VW94].

When clear from the context we often write the formula $A(f_1, ..., f_n)$ as A. The name of the automaton identifies the formulas $f_1, ..., f_n$ nested within it.

Given a formula g, the *models* of the formula is the set $L(g)$ of all infinite words $w \in (2^{\mathcal{PROP}})^\omega$ that satisfy the formula. Given an automaton (either nondeterministic, alternating or 2-way alternating) A with alphabet Σ, the *language* of the automaton A is the set $L(A)$ of all infinite words $w \in \Sigma^\omega$ accepted by A. The *complementary language* is the set $\Sigma^\omega \setminus L(A)$ of all infinite words $w \in \Sigma^\omega$ rejected by A.

Chapter 2

Translating ETL formulas into nondeterministic Büchi automata

In this chapter we translate *ETL* formulas into nondeterministic Büchi automata. We repeat the process three times for ETL_f, ETL_l and ETL_r. First, given a formula g we construct an alternating Büchi automaton $\mathcal{A}_g$ such that $L(\mathcal{A}_g) = L(g)$. Then, we build a nondeterministic Büchi automaton B such that $L(B) = L(\mathcal{A}_g)$.

2.1 Translating finite and looping acceptance ω-automata into alternating Büchi automata

Our goal is given an $\mathrm{ETL}_f(\mathrm{ETL}_l)$ formula g, to construct the alternating automaton $\mathcal{A}_g$ such that an infinite word is a model for g if and only if it is accepted by the automaton $\mathcal{A}_g$. It makes sense to replace the automata connectives in ETL_l and ETL_f by alternating automata and then plug these automata into a bigger alternating automaton that takes care of the boolean structure of the formulas (much like [Var96]). In order to do so for a given finite (looping) acceptance automaton we build two alternating automata. The first recognizing the same language as the finite (looping) automaton and the second recognizing the complementary language. Since nondeterministic automata are a special case of alternating automata, the first transformation is straightforward. Although the automata we are dealing with read infinite objects, their simple acceptance condition makes complementing very easy. We simply take the dual of the automaton [MS87]. For the sake of completeness we include the full construction.

Let $A_f = \langle \Sigma, S, S_0, \rho, F \rangle$ be a finite acceptance ω-automaton. Let $S' = S \cup \{s_0\}$ and assume $s_0 \notin S$. The two alternating automata are:

- $A^a_f = \langle \Sigma, S', s_0, \rho^a_f, \emptyset \rangle$, where
  - $\rho^a_f(s_0, a) = \begin{cases} true & S_0 \cap F \neq \emptyset \\ \bigvee_{s \in S_0} \bigvee_{p \in \rho(s,a)} p & S_0 \cap F = \emptyset \end{cases}$
  - $\rho^a_f(s, a) = \begin{cases} true & s \in F \\ \bigvee_{p \in \rho(s,a)} p & s \notin F \end{cases}$
- $\overline{A}^a_f = \langle \Sigma, S', s_0, \overline{\rho}^a_f, S \rangle$, where:

- $\overline{\rho}^a_f(s_0, a) = \begin{cases} false & S_0 \cap F \neq \emptyset \\ \bigwedge_{s \in S_0} \bigwedge_{p \in \rho(s,a)} p & S_0 \cap F = \emptyset \end{cases}$
- $\overline{\rho}^a_f(s, a) = \begin{cases} false & s \in F \\ \bigwedge_{p \in \rho(s,a)} p & s \notin F \end{cases}$

Note that the transition of $\overline{A}$ uses only conjunctions. Hence it has only one possible run on a word. In this sense it is somewhat deterministic, a fact that is used in the following proofs.

Claim 2.1.1 $L(A^a_f) = L(A_f)$

Proof: An accepting run of A_f on a word w induces an accepting run of A^a_f on the same word (exchange of first state needed) and vice versa. □

Note that it seems as though the alternating automaton is reading one more letter, it reaches the accepting state and only the next transition simplifies to true. The depth of the run tree of the alternating automaton, however, is exactly the length of the run of the nondeterministic automaton. Since the model is infinite and there is always another letter this does not change the language of the automaton.

Claim 2.1.2 $L(\overline{A}^a_f) = \Sigma^\omega \setminus L(A_f)$

Proof: We first show that a word accepted by A_f is rejected by $\overline{A}^a_f$. An accepting run of A_f on a word w reaches an accepting state at some stage. The same run is a path in the tree run of $\overline{A}^a_f$. This path reaches 'false' and $\overline{A}^a_f$ rejects. In the other direction, given a word w rejected by $\overline{A}^a_f$ it is accepted by A_f. As mentioned $\overline{A}^a_f$ has a unique run over w. Since the run of $\overline{A}^a_f$ on w is rejecting a path in this run reaches 'false'. The same path induces an accepting run of A_f on w. □

Similarly for a looping acceptance ω-automaton $A_l = \langle \Sigma, S, S_0, \rho, \emptyset \rangle$, define the following two alternating automata:

- $A^a_l = \langle \Sigma, S', s_0, \rho^a_l, S \rangle$, where
  - $\rho^a_l(s_0, a) = \bigvee_{s \in S_0} \bigvee_{p \in \rho(s,a)} p$
  - $\rho^a_l(s, a) = \bigvee_{p \in \rho(s,a)} p$
- $\overline{A}^a_l = \langle \Sigma, S', s_0, \overline{\rho}^a_l, \emptyset \rangle$, where
  - $\overline{\rho}^a_l(s_0, a) = \bigwedge_{s \in S_0} \bigwedge_{p \in \rho(s,a)} p$
  - $\overline{\rho}^a_l(s, a) = \bigwedge_{p \in \rho(s,a)} p$

Note that an empty disjunction amounts to 'false' and an empty conjunction amounts to 'true'. Thus, if $\rho(s', a') = \emptyset$ then $\rho^a_l = false$ and $\overline{\rho}^a_l = true$. Once again the 'negative' automaton has a unique run over a word.

Claim 2.1.3 $L(A^a_l) = L(A_l)$

Proof: An accepting run of A_l on a word w induces an accepting run of A_l^a on the same word (exchange of first state needed) and vice versa. □

Claim 2.1.4 $L(\overline{A}_l^a) = \Sigma^\omega \setminus L(A_l)$

Proof: An accepting run of A_l on a word w is an infinite run. In the unique tree run of A_l^a on w the same path never reaches 'true' and the automaton rejects. In the other direction a path in the rejecting run of $\overline{A}_l^a$ does not reach 'true'. The same path provides an accepting run for A_l. □

We have built for every finite or looping acceptance automaton two alternating automata with one additional state that accept the same language and the complementary language.

2.2 Negative Normal Form and closure of an ETL formula

Since the transitions of an alternating automaton are of the form $\varphi \in B^+(S)$ negation in the logic presents a problem. As in the translation of temporal logic formulas into automata [GPVW95], negation is dealt with ahead of time. Negations are pushed downwards to apply to automata and propositions only. This is done recursively by:

- Changing $\neg(\alpha \wedge \beta)$ into $(\neg\alpha) \vee (\neg\beta)$
- Changing $\neg(\alpha \vee \beta)$ into $(\neg\alpha) \wedge (\neg\beta)$

By De-Morgan rules the models of the formula do not change. Given a formula g, we denote by $\overline{g}$ the negative normal form of $\neg g$.

The closure of the formula g is intended to serve as the state set of the alternating automaton $\mathcal{A}_g$. We basically follow the definition of the closure in [VW94] but when finding an automaton connective it is replaced by its alternating equivalent (that is, a positive automaton for a positive alternating automaton, and a negated automaton for a negative alternating automaton).

Before defining closure we give the following conventions:

- Identify the formula $\overline{\overline{g}}$ with g.
- Given an alternating automaton $A = \langle \Sigma, S, s_0, \rho, F \rangle$, for each $s \in S$ we define $A_s = \langle \Sigma, S, s, \rho, F \rangle$.

Now, the closure $cl(g)$ of an ETL_f formula g is the minimal set such that:

- $g \in cl(g)$
- if $g_1 \in cl(g)$ then $\overline{g_1} \in cl(g)$
- if $g_1 \wedge g_2 \in cl(g)$ then $g_1, g_2 \in cl(g)$
- if $g_1 \vee g_2 \in cl(g)$ then $g_1, g_2 \in cl(g)$
- if $A^a(g_1, ..., g_n) \in cl(g)$ then $g_1, ..., g_n \in cl(g)$

Note that all elements in the closure are in negative normal form. Negations are applied to automata and propositions only.

We would like to use the alternating automata prepared in the previous section to replace the automata connectives in the closure. We replace all automata connectives (or their negation) by the appropriate alternating automata: For the connective $A(f_1, ..., f_n)$ where $A = \langle \Sigma, S, S_0, \rho, F \rangle$ we prepared the alternating automata $A^a = \langle \Sigma, S', s_0, \rho^a, F \rangle$ and $\overline{A}^a = \langle \Sigma, S', s_0, \overline{\rho^a}, F \rangle$, so we replace $A(f_1, ..., f_n)$ by $A^a_{s_0}$ and $\neg A(f_1, ..., f_n)$ by $\overline{A}^a_{s_0}$. Finally we add to the closure A^a_s and $\overline{A}^a_s$ for every $s \in S'$. After the completion of this phase in all elements in the closure negation applies to propositions only.

The number of elements in $cl(g)$ is at most twice the size of g.

2.3 ETL_f into alternating Büchi automata

We show now how, given an ETL_f formula g, to build an alternating automaton $\mathcal{A}_g$ such that the language of the automaton is the set of models of g. Like the translation of LTL to alternating automata [Var96] we let the transition of the alternating automaton deal with the boolean connectives and plug in the transition of the alternating automata from Section 2.1.

Theorem 2.3.1 *For every ETL_f formula g of length n there exists an alternating Büchi automaton $\mathcal{A}_g$ such that $L(\mathcal{A}_g) = L(g)$ and $\mathcal{A}_g$ has at most $2n$ states.*

In the following construction we use the alternating automata defined in Section 2.1. We have to modify them slightly. Recall that the finite acceptance automaton accepts a word when it reaches an accepting state. Its alternating counterpart reads an extra letter before declaring 'true'. We have to amend this difference (and a similar difference when dealing with negated connectives) by replacing every A^a_s where $s \in F$ with 'true' and replacing every $\overline{A}^a_s$ where $s \in F$ with 'false'. Using this convention we may assume that for all automata connectives $A(f_1, ..., f_n)$ where $A = \langle \Sigma, S, S_0, \rho, F \rangle$ the intersection of F and S_0 is empty, otherwise the formula is identical to 'true'.

We give now the detailed construction. Given an ETL_f formula g, define the following alternating Büchi automaton $\mathcal{A}_g = \langle 2^{\mathcal{PROP}}, cl(g), g, \Delta, \mathcal{F} \rangle$, where the transition function Δ and the acceptance set $\mathcal{F}$ are defined as follows.

- The transition function $\Delta : cl(g) \times 2^{\mathcal{PROP}} \to B^+(cl(g))$ is defined by induction.
  - $\Delta(true, a) = true$
  - $\Delta(false, a) = false$
  - $\Delta(p, a) = \begin{cases} true & p \in a \\ false & p \notin a \end{cases}$
  - $\Delta(\neg p, a) = \begin{cases} false & p \in a \\ true & p \notin a \end{cases}$
  - $\Delta(g_1 \wedge g_2, a) = \Delta(g_1, a) \wedge \Delta(g_2, a)$
  - $\Delta(g_1 \vee g_2, a) = \Delta(g_1, a) \vee \Delta(g_2, a)$

- For $\varphi \in B^+(S)$ define $replace^S_A(\varphi)$ by replacing $q \in S$ by A_q. For example, $replace^S_{A^a}((s \wedge t) \vee q) = (A^a_s \wedge A^a_t) \vee A^a_q$ and $replace^S_{\overline{A}^a}((s \wedge t) \vee q) = (\overline{A}^a_s \wedge \overline{A}^a_t) \vee \overline{A}^a_q$. Recall that for $s \in F$, we identified A^a_s with 'true' and $\overline{A}^a_s$ with 'false'. Now,
  $\Delta(A^a(g_1, ..., g_n), a) = \bigvee_{i=1}^n [\Delta(g_i, a) \wedge replace^S_{A^a}(\rho(s_0, a_i))]$
  We check that g_i in fact holds ($\Delta(g_i, a)$) and continue the computation of A^a. One computation path has to reach a state in F.
- $\Delta(\overline{A}^a(g_1, ..., g_n), a) = \bigwedge_{i=1}^n [\Delta(\overline{g}_i, a)] \vee replace^S_{\overline{A}^a}(\overline{\rho}(s_0, a_i))$
  We check that either g_i does not hold (recall that $\overline{g}_i$ is the negative normal form of g_i) or the computation of $\overline{A}^a$ has to continue. All enabled paths have to either reach a dead end or run forever without reaching an accepting state.

- The acceptance set is $\mathcal{F} = \{\overline{A}^a_s | \overline{A}^a = \langle \Sigma, S', s, \overline{\rho}^a, S \rangle \in cl(g) \text{ and } s \in S\}$.

  This way positive automata are checked to reach an accepting state and 'vanish'. Negative automata on the other hand are allowed to (and should) run forever.

In the definition of $\Delta(\overline{A}, a)$ we used the negation of formulas g_i. This is the only reason to include negation of formulas in the closure.

Claim 2.3.2 $L(\mathcal{A}_g) = L(g)$

Proof: We prove by induction on the structure of the formula that for all subformulas $f \in cl(g)$, we have $L(\mathcal{A}_f) = L(f)$.

- For propositions and boolean quantifiers the proof is not different from the classical proof.

- Consider an automaton connective $A(f_1, ..., f_n)$, $A = \langle \Sigma, S, S_0, \rho, F \rangle$ has the alternating equivalent $A^a = \langle \Sigma, S', q_0, \rho_a, \emptyset \rangle$.

  A word $w = w_0 w_1 ...$ is a model for $A(f_1, .., f_n)$ iff there is an accepting run $\sigma = s_0, s_1, ..., s_m$ of A where for all $0 \leq k < m$, there is some $a_{j_k} \in \Sigma$ such that $(w, k) \models f_{j_k}$, $s_{k+1} \in \rho(s_k, a_{j_k})$, and $s_m \in F$.

  For the formulas $f_1, ..., f_n$ we can use the induction assumption: $(w, j) \models f_i$ iff $w_{\geq j} \in L(\mathcal{A}_{f_i})$. Hence if $(w, k) \models f_{j_k}$ there is an accepting run of $\mathcal{A}_{f_{j_k}}$ on $w_{\geq k}$. We modify the path run of $A(f_1, ..., f_n)$ on w into a tree by appending to the path the runs of $\mathcal{A}_{f_{j_k}}$. We can also remove the node labeled by s_m since in the run of $\mathcal{A}_{A(f_1,...,f_n)}$ we know that $replace^S_A(\rho(s_{m-1}, a_{j_{m-1}})) = true$, hence under the $m-1$ node in the path we append only the run of $\mathcal{A}_{f_{j_{m-1}}}$.

  Obviously the run answers the demands of the transition of $\mathcal{A}$ and the only path labeled by the automaton A is finite.

  If $w \in L(\mathcal{A})$ there is a sequence $A^a_{s_0}, A^a_{s_1}, ..., A^a_{s_{m-1}}$ such that $A^a_{s_{k+1}} \models replace^S_{A_a}(\rho_a(A_{s_k}, a_j))$ for some j and $\Delta(f_j, w_k)$ results in an accepting computation. There exists j such that $replace^S_{A^a}(\rho(s_{m-1}, a_j)) = true$ and $\Delta(f_j, w_{m-1})$ results in an accepting computation. Obviously $w \models A(f_1, ..., f_n)$.

- Consider an automaton connective $\neg A(f_1, ..., f_n)$, $A = \langle \Sigma, S, S_0, \rho, F \rangle$ has the alternating complement $\overline{A}^a = \langle \Sigma, S', q_0, \overline{\rho}_a, S \rangle$.

It cannot be the case that the same word is a model for $A(f_1, ..., f_n)$ and it is accepted by $\mathcal{A}_{\neg A}$. If this is the case there exists an accepting run $\sigma = s_0, ..., s_m$ of $A(f_1, ..., f_n)$ on w and a tree run (T, V) of $\mathcal{A}_{\neg A}$ on w. According to the structure of $\mathcal{A}$ we deduce that in level $m-1$ in the tree there is a node labeled by $\overline{A}^a_{s_{m-1}}$. Since $s_m \in \rho(s_{m-1}, a_j)$, $(w, m-1) \models f_j$ and $w_m \in F$ we know that $replace^S_A(\overline{\rho}(s_{m-1}, a_j)) = false$ and the run of $\mathcal{A}$ has to be rejecting. Therefore $L(\mathcal{A}_{\neg A}) \cap models(A(f_1, ..., f_n)) = \emptyset$.

On the other hand if $w \notin models(A(f_1, ..., f_n))$ then for every run of $A(f_1, ..., f_n)$ on w

- either the run $s_0, s_1, ...$ is infinite and never reaches an accepting state: for all $k \geq 0$ there exists some $a_j \in \Sigma$ such that $(w, k) \models f_j$ and $s_{k+1} \in \rho(s_k, a_j)$ and $s_k \notin F$.
- or the run $s_0, s_1, ..., s_m$ is finite, never reaches an accepting state and gets to a point where none of the formulas $f_1, ..., f_n$ hold: for all $0 \leq k < m$, $s_k \notin F$ and there exists some $a_j \in \Sigma$ such that $(w, k) \models f_j$ and $s_{k+1} \in \rho(s_k, a_j)$ and for all $a_j \in \Sigma$, $(w, m) \not\models f_j$.

We can build the run of $\mathcal{A}_{\neg A}$ by induction. Label the root by $\overline{A}^a_{s_0}$. For all $a_j \in \Sigma$ if $w \models f_j$, it must be the case that no a_j successor of s_0 is a member of F (i.e. $\rho(s_0, a_j) \cap F = \emptyset$) because otherwise w is a model of $A(f_1, ..., f_n)$ contrary to the assumption. Hence we add $|\rho(s_0, a_j)|$ successors to the root and label them by $\overline{A}^a_t$ for $t \in \rho(s_0, a_j)$. If $w \not\models f_j$ we append the run tree of $\mathcal{A}_{\overline{f_j}}$ on w under the root (unifying the roots).

For a leaf x in the tree, if x is labeled by any proper subformula of $A(f_1, ..., f_n)$ then it was appended as a part of a complete run tree and we are ensured that the transition $\Delta(V(x), w_{|x|}) = true$. If it is labeled by $\overline{A}^a_t$ for some $t \in S$ we can repeat the process applied to the root. Since we assumed w is not a model of $A(f_1, ..., f_n)$, no successor is a member of F. We know that $replace^S_A(\overline{\rho}(t, a)) \neq false$ since no successor can be in F. The transition of Δ is satisfied by the resulting tree and we are done.

□

2.4 ETL_l into alternating Büchi automata

Similar to the previous section, given an ETL$_l$ formula g we build an alternating automaton such that the language of the automaton is the set of models of g. The construction of $\mathcal{A}_g$ is very similar to the ETL_f case.

Theorem 2.4.1 *For every ETL$_l$ formula g of length n there exists an alternating Büchi automaton $\mathcal{A}_g$ such that $L(\mathcal{A}_g) = L(g)$ and $\mathcal{A}_g$ has at most $2n$ states.*

Unlike the case of ETL$_f$, because of the infinite nature of ETL$_l$ there is no need to give special attention to the time we identify the entry into the accepting set. We describe part of the transition function dealing with automata connectives and prove the construction is correct. So $\mathcal{A}_g = \langle 2^{\mathcal{PROP}}, cl(g), g, \Delta, \mathcal{F} \rangle$, where the transition function Δ and the acceptance set $\mathcal{F}$ are defined as follows.

- The transition function Δ is defined as in for ETL$_f$. We recall part of the definition.

- $\Delta(A^a(g_1,...,g_n),a) = \bigvee_{i=1}^{n}[\Delta(g_i,a) \wedge replace^S_{A^a}(\rho_a(s_0,a_i))]$

  We check that g_i holds ($\Delta(g_i,a)$) and continue the computation of A^a. There has to be an infinite path.

- $\Delta(\overline{A}^a(g_1,...,g_n),a) = \bigwedge_{i=1}^{n}[\Delta(\overline{g_i},a) \vee replace^S_{\overline{A}^a}(\overline{\rho}_a(s_0,a_i))]$

  We check that either g_i does not hold or the computation of $\overline{A}^a$ continues. All enabled paths have to reach a dead end of some sort.

- The acceptance set is $\mathcal{F} = \{A^a_s | A^a_s = \langle \Sigma, S, s, \rho^a, F\rangle \in cl(g) \text{ and } s \in S\}$.

  When dealing with looping acceptance automata, unlike finite acceptance, the positive automata may appear on infinite paths but all negative automata must appear only on finite paths.

Claim 2.4.2 $L(\mathcal{A}_g) = L(g)$

Proof: We prove by induction on the structure of the formula that for all subformulas $f \in cl(g)$, we have $L(\mathcal{A}_f) = L(f)$.

- Consider an automaton connective $A(f_1,...,f_n)$, $A = \langle \Sigma, S, S_0, \rho, S\rangle$ has the alternating equivalent $A^a = \langle \Sigma, S', s_0, \rho_a, S\rangle$.

  A word $w = w_0w_1...$ is a model for $A(f_1,...,f_n)$ iff there is an accepting run $\sigma = s_0, s_1, ...$ of A where for all $k \geq 0$ there is some $a_{j_k} \in \Sigma$ such that $(w,k) \models f_{j_k}$ and $s_{k+1} \in \rho(s_k, a_{j_k})$.

  For the formulas $f_1,...,f_n$ we can use the induction assumption: $(w,j) \models f_i$ iff $w_{\geq j} \in L(\mathcal{A}_{f_i})$. We modify the path run of $A(f_1,...,f_n)$ on w into a tree by appending to the path the runs of $\mathcal{A}_{f_{j_k}}$ on w_k.

  Obviously the run answers the demands of the transition of $\mathcal{A}$ and the only path labeled by the automaton A is infinite.

  If $w \in L(\mathcal{A})$ then there is an infinite sequence $A^a_{q_0}, A^a_{q_1}, ...$ such that $A^a_{s_{k+1}} \models replace^S_{A^a}(\rho_a(A_{s_k}, a_j))$ for some j and $\Delta(f_j, w_k)$ results in an accepting computation. The same holds for A therefore $w \models A(f_1,...,f_n)$

- Consider an automaton connective $\neg A(f_1,...,f_n)$, $A = \langle \Sigma, S, S_0, \rho, F\rangle$ has the alternating complement $\overline{A}^a = \langle \Sigma, S, q_0, \overline{\rho}_a, \emptyset\rangle$.

  It cannot be the case that the same word is a model for $A(f_1,...,f_n)$ and it is accepted by $\mathcal{A}_{\neg A}$. If this is the case there exists an accepting run $\sigma = s_0, s_1, ...$ of $A(f_1,...,f_n)$ on w and a tree run of $\mathcal{A}_{\neg A}$ on w. According to the structure of $\mathcal{A}$ we know that all paths labeled by $\overline{A}^a_t$ for $t \in S$ have to be finite. We show by induction that $\overline{A}^a_{s_i}$ appears in level i in the tree. The root is labeled by $\overline{A}^a_{s_0}$. Given a node x in level i in the tree labeled by $\overline{A}^a_{s_i}$ we know that there exists some $a_j \in \Sigma$ s.t. $(w, i+1) \models f_j$ and $s_{i+1} \models \rho(s_i, a_j)$. Hence $\Delta(\overline{f_j}, w_i)$ cannot result in an accepting run tree and there has to be a node under x labeled by $\overline{A}^a_{s_{i+1}}$. Therefore $L(\mathcal{A}_{\neg A}) \cap L(A(f_1,...,f_n)) = \emptyset$.

  On the other hand if $w \notin L(A(f_1,...,f_n))$ then every run of $A(f_1,...,f_n)$ on w is rejecting

  - either because $\rho(s_m, a_j) = \emptyset$ for all $a_j \in \Sigma$ such that $(w,m) \models f_j$
  - or because for all $a_j \in \Sigma$, $(w,m) \not\models f_j$

We can build the run of $\mathcal{A}_{\neg A}$ by induction. Label the root by $\overline{A}^a_{s_0}$. For all $a_j \in \Sigma$ such that $w \models f_j$, if there are no a_j successors of s_0 (i.e. $\rho(s_0, a_j) = \emptyset$) we are done. Otherwise we add $|\rho(s_0, a_j)|$ successors to the root and label them by $\overline{A}^a_t$ for $t \in \rho(s_0, a_j)$. If $w \not\models f_j$ we append the run tree of $\mathcal{A}_{\overline{f_j}}$ on w under the root (unifying the roots).

For a leaf x in the tree, if x is labeled by any proper subformula of $A(f_1, ..., f_n)$ then it was appended as a part of a complete run tree and we are ensured that the transition $\Delta(V(x), w_{|x|}) = true$. If it is labeled by $\overline{A}^a_t$ for some $t \in S$ we can repeat the process applied to the root.

There cannot be an infinite path in the run of $\mathcal{A}_{\neg A}$ labeled by $\overline{A}^a$. Such a path can be converted into a run of $A(f_1, ..., f_n)$ on w contradicting the assumption. Other infinite paths are labeled by other automata from a certain point onward. In this case those infinite paths were added as a part of an infinite accepting run tree and visit the acceptance set $\mathcal{F}$ infinitely often.

□

2.5 From Alternating Büchi automata to nondeterministic Büchi automata

Converting alternating automata into nondeterministic automata involves some sort of subset construction. The states of the resulting automaton are sets of formulas. Intuitively all formulas appearing in the state have to be checked to hold over the model. The special structure of logic enables two approaches:

- A formula not appearing in the state is false in this state and its falseness should be checked. We call this approach the *tight* approach.
- A formula not appearing in the state is not interesting. We call this approach the *loose* approach.

A formula is either true or false. Hence, when using the tight approach a formula either belongs to the state or does not belong to it. Using the loose approach, a formula either belongs to the state, or its negation belongs to the state or, not caring about this formula, none of the two belongs. Obviously, it cannot be the case where both the formula and its negation appear in the same set. There are advantages and disadvantages for both approaches (see [GPVW95, DFV99]).

Theorem 2.5.1 *For every ETL_f or ETL_l formula g of length n there exists a tight (loose) nondeterministic Büchi automaton B such that $L(B) = L(g)$ and B has at most 3^n (4^n) states.*

The simplest approach to converting alternating Büchi automata into nondeterministic Büchi automata is to use the construction in [MH84]. Given an alternating Büchi automaton $A = \langle \Sigma, S, s_0, \rho, F \rangle$ they propose $B = \langle \Sigma, 2^S \times 2^S, (\{s_0\}, \emptyset), \rho', 2^S \times \{\emptyset\} \rangle$, where ρ' is defined, for all $(P, Q) \in 2^S \times 2^S$ and $a \in \Sigma$ as follows.

- If $Q \neq \emptyset$ then $\rho'((P, Q), a) =$

$$\left\{ (P', Q' \setminus F) \;\middle|\; \begin{array}{l} P' \text{ satisfies } \bigwedge_{p \in P} \rho(p, a) \\ Q' \subseteq P', \text{and} \\ Q' \text{ satisfies } \bigwedge_{p \in Q} \rho(p, a) \end{array} \right\}$$

- If $Q = \emptyset$ the $\rho'((P,Q),a) =$

$$\{(P', P' \setminus F) | P' \text{ satisfies } \bigwedge_{p \in P} \rho(p,a)\}$$

This way the first component in the state of B follows all the paths in a run tree of A in the same time. The second component collects only paths that owe a visit to the acceptance set F. Once the second component is empty (all paths visited F at least once) it is refilled with the new level in the run tree of A. If the second component is empty infinitely often we are ensured that every path in the tree of A visited F infinitely often. As noted by Isli [Isl96], all reachable states are of the form $(P,Q) \in 2^S \times 2^S$ where $Q \subseteq P$. Hence we can replace the state set by 3^S, where 0 indicates not appearing, 1 indicates appearing only in the first component and 2 indicates appearing in both components. The second component in the states of B is often referred to as the *book-keeping* component.

This construction yields for an alternating automaton with n states a nondeterministic automaton with 3^n states. Given an ETL formula g of length n, the alternating automaton $\mathcal{A}_g$ has $2n$ states. Therefore the final nondeterministic automaton has 3^{2n} states.

This result can be improved using either the tight approach or the loose approach. We create the *reduced closure* of the formula. Let $rcl(g)$ be a subset of g such that for every formula $f \in cl(g)$ either $f \in rcl(g)$ or $\overline{f} \in rcl(g)$ and it is not the case that $f \in rcl(g)$ and $\overline{f} \in rcl(g)$. Furthermore, all propositions and automata connectives appear in the reduced closure in their positive form (i.e. for all $p \in \mathcal{PROP}$, we have $p \in rcl(g)$ and for all $A_s \in cl(g)$, we have $A_s \in rcl(g)$). In the following we reduce the number of states from 9^n to either 3^n, using the tight approach, or 4^n, using the loose approach. We use the following observation.

- Take the run tree of $\mathcal{A}_g$ where g is an ETL_f formula. If in the run tree of $\mathcal{A}_g$ appears a node labeled by a negated automaton connective, there might be an infinite path under that node labeled by the same negated automaton in different states. No other condition is imposed on this infinite path. All these states (negated automata) are members of the accepting set and they never appear in the book-keeping component. On the other hand automata connectives that are not negated have to be checked to make sure they do not run forever.

- For ETL_l formulas the opposite is true. Thus, negated automata connectives have to be checked to have no infinite paths and non-negated automata may have infinite paths.

We start with the tight approach. We describe only the ETL_f construction, the construction for ETL_l is similar. Given a subset U of $cl(g)$ and the set $rcl(g)$ we say that a formula $f \in rcl(g)$ appears *positive* in U if $f \in U$ and appears *negative* in U if $\overline{f} \in U$. In the tight approach we use states from $\{-1,1,2\}^{rcl(g)}$. Each state $P \in \{-1,1,2\}^{rcl(g)}$ represents a subset U of $cl(g)$. For a formula $f \in rcl(g)$, if f's coordinate in P is -1 it indicates that $\overline{f} \in U$, if f's coordinate in P is 1 it indicates that $f \in U$, and if f's coordinate in P is 2 it indicates that $f \in U$ and that the nondeterministic automaton is following f also in the book-keeping component.

In order to simplify notations, the states of the nondeterministic automaton consist of two subsets of $2^{rcl(g)}$. Converting our automaton to an automaton using $\{-1,1,2\}^{rcl(g)}$, as above, is straightforward.

We first confine the set $2^{rcl(g)}$ to the set of all consistent subsets: if a disjunction is a member of the set, one of the disjuncts has to be in the set as well, and if a conjunction is a member of the

set, both conjuncts have to be in the set.

$$cons(2^{rcl(g)}) = \left\{ P \in 2^{rcl(g)} \;\middle|\; \begin{array}{l} \forall(f_1 \wedge f_2) \in rcl(g),\ f_1 \wedge f_2 \in P \iff f_1 \in P \text{ and } f_2 \in P \\ \forall(f_1 \vee f_2) \in rcl(g),\ f_1 \vee f_2 \in P \iff f_1 \in P \text{ or } f_2 \in P \end{array} \right\}$$

Given $\mathcal{A}_g = \langle 2^{\mathcal{PROP}}, cl(g), g, \Delta, \mathcal{F} \rangle$, we build the nondeterministic automaton $B = \langle 2^{\mathcal{PROP}}, S \times S, S_0 \times \{\emptyset\}, \Delta', S \times \{\emptyset\} \rangle$, where

- $S = cons(2^{rcl(g)})$
- $S_0 = \{t \in S \mid g \in t \text{ or } \overline{g} \notin t\}$, the initial states are the ones for which g is checked to be true.
- The transition function Δ' is defined for all $(P, Q) \in S \times S$ and $a \subseteq \mathcal{PROP}$ as follows.
  - If $Q = \emptyset$, then $(P', Q') \in \Delta'((P, Q), a)$ iff all the following conditions hold.
    - For all $p \in \mathcal{PROP}$, we have $p \in P$ iff $p \in a$.
    - For all automata connectives $A(f_1, ..., f_n)$ with $A = \langle \Sigma, S, S_0, \rho, F \rangle$.
      - If $A_s \in P$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or $\overline{f_j} \notin P$ and either $t \in F$ or $A_t \in P'$.
      - If $A_s \notin P$, then for all A_t such that $t \in \rho(s, a_j)$ where $f_j \in P$ or $\overline{f_j} \notin P$, we have $A_t \notin P'$.
    - $Q' = \{A_s | A_s \in P'\}$.
  - If $Q \neq \emptyset$ then $(P', Q') \in \Delta'((P, Q), a)$ iff all the following conditions hold.
    - For all $p \in \mathcal{PROP}$, we have $p \in P$ iff $p \in a$.
    - For all automata connectives $A(f_1, ..., f_n)$ with $A = \langle \Sigma, S, S_0, \rho, F \rangle$.
      - If $A_s \in P$ and $A_s \notin Q$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or $\overline{f_j} \notin P$ and either $t \in F$ or $A_t \in P'$.
      - If $A_s \notin P$, then for all A_t such that $t \in \rho(s, a_j)$ where $f_j \in P$ or $\overline{f_j} \notin P$, we have $A_t \notin P'$.
      - If $A_s \in P$ and $A_s \in Q$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or $\overline{f_j} \notin P$ and either $t \in F$ or $A_t \in P'$ and $A_t \in Q'$.

  The transition function requires that every positive automaton is followed by one possible successor and every negative automaton is followed by all possible successors. The book-keeping component makes sure that all paths of positive automata are finite.

Claim 2.5.2 $L(B) = L(\mathcal{A}_g)$

Proof: The proof is quite straightforward. Proving that $L(B)$ is a subset of $L(\mathcal{A}_g)$, we divide the run of the nondeterministic automaton B into a tree run of $\mathcal{A}_g$, the acceptance condition of B makes sure that no path in the tree, labeled with positive automata, is infinite. Proving that $L(\mathcal{A}_g)$ is a subset of $L(B)$, we build a Hintikka Sequence for the word and use the alternating automaton to prove that it satisfies the transition function of B and that the book-keeping component is empty infinitely often. We now dive into the details:

Suppose $w \in L(B)$. Then there exists an accepting run $(P_0, Q_0), (P_1, Q_1), ...$ of B on w. We build by induction a run tree (T, V) of $\mathcal{A}_g$ on w such that the set of labels of the nodes in level i

in the tree is a subset of P_i or formally for all nodes $x \in T$ such that $|x| = i$ either $V(x) \in P_i$ or $\overline{V(x)} \notin P_i$. We start by $V(\epsilon) = g$, since $(P_0, Q_0) \in S_0,\ g \in P_0$ or $\overline{g} \notin P_0$. Given a node $x \in T$ the label of x is either an automaton in some state or a proposition.

- If the label is a (negated) proposition $V(x) \in \mathcal{PROP}$ then by induction assumption it is in $P_{|x|}$ and we can conclude that $V(x) \in w_{|x|}$ ($V(x) \notin w_{|x|}$).
- If the label is some automaton $V(x) = A_s$ where the connective is $A(f_1, ..., f_n)$ and A_s does not appear in the book-keeping component, $A_s \notin Q_{|x|}$, then there has to be some $t \in \rho(s, a_j)$ such that $f_j \in P_{|x|}$ or $\overline{f_j} \notin P_{|x|}$ and either $t \in F$ and then we are done or $A_t \in P_{|x|+1}$. In this case we add a successor to x in T and label it $V(x) = A_t$. We take apart f_j (or $\overline{f_j}$) and get its propositional and automata components, propositions are fulfilled (they are fulfilled in the run of B) and automata parts are handled as if labeling x.
- If the label is some automaton $V(x) = A_s$ where the connective is $A(f_1, ..., f_n)$ and A_s appears in the book-keeping component, $A_s \in Q_{|x|}$, then if A_s has a successor in the accepting set (i.e. for some $f_j \in P_{|x|}$ or $\overline{f_j} \notin P_{|x|}$ there is $t \in \rho(s, a_j)$ such that $t \in F$) then we only handle the propositional and automata requirements for f_j. Otherwise we follow the path inside the book-keeping component in a similar way to the previous item and handle f_j (or $\overline{f_j}$).
- If the label is some negative automaton $V(x) = \overline{A}_s$ where the connective is $A(f_1, ..., f_n)$ then for all t such that $t \in \rho(s, a_j)$ and $f_j \in P_{|x|}$ or $\overline{f_j} \notin P_{|x|}$ we add successors to x and label them $\overline{A}_t$. For $f_1, ..., f_n$, if f_j appears in $P_{|x|}$ it is satisfied and if f_j does not appear in $P_{|x|}$, take $\overline{f_j}$ apart and handle its components just like before.

The resulting tree is a run tree of $\mathcal{A}_g$. We have to make sure it is accepting.

Assume by contradiction that there is an infinite path $x_0, x_1, ...$ labeled by positive automata. From the construction of the run tree, if the label of x_i is some automaton A_s then the label of x_{i+1} is either the same automaton in another state or an automaton that is nested inside the first. The level of nesting is bounded hence there exists a point i in the path beyond which all the labels belong to the same automaton connective. Since in the run of B the book-keeping component Q is empty infinitely often there is a point $j > i$ such that $Q_j = \emptyset$. Hence the label of x_{j+1} is found in Q_{j+1}. From the construction of the run tree we can deduce that for all $k > j$ the label of x_k is found in Q_k. Since $Q_l = \emptyset$ for infinitely many ls this is a contradiction.

Suppose $w \in L(\mathcal{A}_g)$ then there exists an accepting run tree (T, V) of $\mathcal{A}_g$ on w. Furthermore from the previous parts for every formula f in the closure of g we know that $(w, i) \models f \iff w_{\geq i} \in L(\mathcal{A}_f)$. Hence if a formula f is true at point i of the sequence $(w, i) \models f$, then there exists an accepting run tree (T_f^i, V_f^i) of $\mathcal{A}_f$ on the word $w_{\geq i}$. In particular (T_g^0, V_g^0) is the run of $\mathcal{A}_g$ on w.

We construct the run of B in two stages first we construct the Hintikka sequence that provides for the first component of every ordered pair. Then we complete the second component - the book-keeping component. For the Hintikka sequence we take all the formulas that are true at the time $P_i = \{f \in rcl(g) | (w, i) \models f\}$. Obviously, for every formula f in P_i there exists an accepting run (T_f^i, V_f^i) and for every formula f not in P_i there exists an accepting run $(T_{\overline{f}}^i, V_{\overline{f}}^i)$. This is sufficient to prove that the sequence $P_0, P_1, ...$ is a projection of a run of B on the first component of $S \times S$. Obviously for all i, P_i is consistent and if some automaton connective $A_s(f_1, ..., f_n)$ appears in P_i then either s has an accepting state reachable from it or we can take from the run $(T_{A_s}^i, V_{A_s}^i)$ the element A_t appearing in level 1 of the tree and we know that it is satisfied at time $i + 1$. Similarly

if an automaton connective does not appear in a state all the possible successors do not appear in the following state.

We are left with the 'acceptance' part of the run of B. This is built by induction from empty set to empty set. The first state Q_0 is empty by definition. Given Q_i empty in the run we know that Q_{i+1} holds all the positive automata held in P_{i+1}. Denote $Q_{i+1} = \{A^1_{s_1}, ..., A^p_{s_p}\}$. For every one of these automata there is an accepting run $(T^{i+1}_{A^j_{s_j}}, V^{i+1}_{A^j_{s_j}})$. Since the paths with positive automata in these trees are finite we unite the positive successors of $A^1_{s_1}, ..., A^p_{s_p}$ into the sequence of Qs. Obviously the Qs are subsets of the true formulas. We can gather that for some $l > i$, the set Q_l is empty again. □

In the loose approach, we describe only the ETL_l construction, the construction for ETL_f is similar. We reduce the state set to $\{-2,-1,0,1\}^{rcl(g)}$. Given a subset $U \subseteq cl(g)$, the state $P \in \{-1,0,1,2\}^{rcl(g)}$ represents it. For a formula $f \in rcl(g)$ if f's coordinate in P is -1 it indicates that $\overline{f} \in U$, if f's coordinate in P is 0 it indicates that $\overline{f} \notin U$ and $f \notin U$, if f's coordinate in P is 1 it indicates that $f \in U$ and if f's coordinate in P is 2 it indicates that $f \in U$ and that the nondeterministic automaton is following f also in the book-keeping component. For simplicity of notation we use a separate book-keeping component.

We confine the set $\{-1,0,1\}^{rcl(g)}$ to the set of consistent subsets. Given a set $P \in \{-1,0,1\}^{rcl(g)}$, we abuse notation and write $f \in P$ for $P_f = 1$ (i.e. f's coordinate in P equals 1), $f \overline{\in} P$ if $P_f = -1$ and $f \notin P$ if $P_f = 0$.

$$cons(\{-1,0,1\}^{rcl(g)}) = \left\{ P \;\middle|\; \begin{array}{l} \forall(f_1 \wedge f_2) \in rcl(g),\ f_1 \wedge f_2 \in P \Rightarrow f_1 \in P \text{ and } f_2 \in P \\ \forall(f_1 \vee f_2) \in rcl(g),\ f_1 \vee f_2 \in P \Rightarrow f_1 \in P \text{ or } f_2 \in P \\ \forall(f_1 \wedge f_2) \in rcl(g),\ f_1 \wedge f_2 \overline{\in} P \Rightarrow f_1 \overline{\in} P \text{ or } f_2 \overline{\in} P \\ \forall(f_1 \vee f_2) \in rcl(g),\ f_1 \vee f_2 \overline{\in} P \Rightarrow f_1 \overline{\in} P \text{ and } f_2 \overline{\in} P \end{array} \right\}$$

Given the alternating Büchi automaton $\mathcal{A}_g = \langle 2^{\mathcal{PROP}}, cl(g), g, \Delta, \mathcal{F} \rangle$ we build the following nondeterministic Büchi automaton $B = \langle 2^{\mathcal{PROP}}, S \times S, S_0 \times \{\emptyset\}, \Delta, S \times \{\emptyset\} \rangle$, where

- $S = cons(\{-1,0,1\}^{rcl(g)})$
- $S_0 = \{t \in S | g \in t \text{ or } \overline{g} \overline{\in} t\}$. The initial states are the ones for which g is checked to be true.
- The transition function Δ' is defined for all $(P,Q) \in S \times S$ and $a \subseteq \mathcal{PROP}$ as follows.
  - If $Q = \emptyset$, then $(P', Q') \in \Delta'((P,Q), a)$ iff all the following conditions hold.
    - For all $p \in \mathcal{PROP}$, we have $p \in P$ implies $p \in a$, and $p \overline{\in} P$ implies $p \notin a$.
    - For all automata connectives $A(f_1, ..., f_n)$ with $A = \langle \Sigma, S, S_0, \rho, F \rangle$.
      - If $A_s \in P$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or $\overline{f_j} \overline{\in} P$, and $A_t \in P'$.
      - If $A_s \overline{\in} P$, then for all A_t such that $t \in \rho(s, a_j)$ where $f_j \in P$ or $\overline{f_j} \overline{\in} P$ or ($f_j \notin P$ and $\overline{f_j} \notin P$), we have $A_t \overline{\in} P'$.
    - $Q' = \{A_s | A_s \overline{\in} P'\}$.
  - If $Q \neq \emptyset$, then $(P', Q') \in \Delta'((P,Q), a)$ iff all the following conditions hold.

* For all $p \in \mathcal{PROP}$, we have $p \in P$ implies $p \in a$, and $p \overline{\in} P$ implies $p \notin a$.
* For all automata connectives $A(f_1, ..., f_n)$ with $A = \langle \Sigma, S, S_0, \rho, F \rangle$.
  - If $A_s \in P$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or $\overline{f_j} \overline{\in} P$, and $A_t \in P'$.
  - If $A_s \overline{\in} P$ and $A_s \notin Q$, then for all A_t such that $t \in \rho(s, a_j)$ where $f_j \in P$ or $\overline{f_j} \overline{\in} P$ or ($f_j \notin P$ and $\overline{f_j} \notin P$), we have $A_t \overline{\in} P'$.
  - If $A_s \overline{\in} P$ and $A_s \overline{\in} Q$, then for all A_t such that $t \in \rho(s, a_j)$ where $f_j \in P$ or $\overline{f_j} \overline{\in} P$ or ($f_j \notin P$ and $\overline{f_j} \notin P$), we have $A_t \overline{\in} P'$ and $A_t \overline{\in} Q'$.

The transition function requires that every positive automaton is followed by some successor and every negative automaton is followed by all possible successors. For the subformulas $f_1, ..., f_n$ of the automaton connective $A(f_1, ..., f_n)$, if in the current state P we do not care about a formula f_j (both the formula and its negation do not appear in the state), we assume that it is correct.

Claim 2.5.3 $L(B) = L(\mathcal{A}_g)$

Proof: Given a run of the nondeterministic automaton B, we turn it into a run of A and vice versa.

Suppose $w \in L(B)$ then there exists an accepting run $(P_0, Q_0), (P_1, Q_1), \ldots$ of B on w. We build by induction a run tree (T, V) of $\mathcal{A}_g$ on w such that the set of labels of the nodes in level i in the run tree is a subset of P_i. Formally for all nodes $x \in T$ such that $|x| = i$ either $V(x) \in P_i$ or $\overline{V(x)} \overline{\in} P_i$. We start with $V(\epsilon) = g$, since $P_0 \in S_0$ either $g \in P_0$ or $\overline{g} \overline{\in} P_0$. Given a node $x \in T$ the label of x is either an automaton in some state or a proposition.

- If the label is a (negated) proposition then by the induction assumption it is in $P_{|x|}$ (not in $P_{|x|}$) and $V(x) \in w_{|x|}$ ($\overline{V(x)} \overline{\in} w_{|x|}$).
- If the label is some automaton $V(x) = A_s$ where the connective is $A(f_1, ..., f_n)$ (positive automata do not appear in the book-keeping component), then there has to be some $t \in \rho(s, a_j)$ such that $f_j \in P_{|x|}$ or $\overline{f_j} \overline{\in} P_{|x|}$ and $A_t \in P_{|x|+1}$. We add a successor $x \cdot c$ to x in T and label it $V(x \cdot c) = A_t$.
- If the label is some negative automaton $V(x) = \overline{A}_s$ where the connective is $A(f_1, ..., f_n)$ and $\overline{A}_s$ does not appear in the book-keeping component. For every formula f_i:
  - If we do not care about f_i ($f_i \notin P_{|x|}$ and $\overline{f_i} \notin P_{|x|}$) then if $\rho(s, a_j)$ is empty we are done. Otherwise for every $t \in \rho(s, a_j)$, $\overline{A}_t$ appears in $P_{|x|+1}$. We add a successor to x and label it by $\overline{A}_t$.
  - If f_i is positive ($f_i \in P_{|x|}$ or $\overline{f_i} \overline{\in} P_{|x|}$) we handle $\overline{A}_s$ just like when we do not care about f_j.
  - If f_i is negative ($\overline{f_i} \in P_{|x|}$ or $f_i \overline{\in} P_{|x|}$) then as $P_{|x|}$ is consistent, all subformulas of f_i are cared about. We handle these subformulas as if labeling x.
- If the label is some negative automaton $V(x) = \overline{A}_s$ where the connective is $A(f_1, ..., f_n)$ and $\overline{A}_s$ appears in the book-keeping component. We handle it just like we handled a negative automaton not appearing in the book-keeping component but follow its descendents in $Q_{|x|+1}$ rather than in $P_{|x|+1}$.

Assume by way of contradiction that the resulting tree is not accepting. In this case there is an infinite path of negative automata. Just like in the previous proof of the tight case this path will finally get trapped in the book-keeping component. Since the book-keeping component is empty infinitely often, this is a contradiction.

Assume $w \in L(\mathcal{A}_g)$. In order to show that w is accepted also by B we can use the previous proof for the tight nondeterministic automaton. We simply use B as a tight automaton, disallowing the dont care state. Thus the proof is a simple variant of the proof in the tight case (for ETL_f) and we omit it. □

2.6 From ETL_r formulas to nondeterministic Büchi automata

In this section we construct for an ETL_r formula g a nondeterministic Büchi automaton B such that $L(B) = B(g)$. The work in this section is very similar to the work in the previous sections. Given a nondeterministic Büchi automaton we show how to construct an alternating Büchi automaton accepting the same language and an alternating Büchi automaton accepting the complementary language. Given an ETL_r formula g, we build the alternating Büchi automaton $\mathcal{A}_g$ such that $L(g) = L(\mathcal{A}_g)$. Finally, we transform $\mathcal{A}_g$ to a nondeterministic Büchi automaton.

2.6.1 From nondeterministic Büchi automata to alternating Büchi automata

As in the first part, we start by building two alternating automata. Given a nondeterministic Büchi automaton we build an alternating automaton that accepts the same language and an alternating automaton that accepts the complementary language. We use the constructions given in [KV97] and [Tho98]. We use the following notations $S' = S \cup \{s_0\}$, $[k] = \{0, 1, ..., k\}$ and $Odd(P) = \{i \in P | i \text{ is odd}\}$.

Given a nondeterministic Büchi automaton $A = \langle \Sigma, S, S_0, \rho, F \rangle$, we define

- $A^a = \langle \Sigma, S', s_0, \rho^a, F \rangle$, where
  - $\rho^a(s_0, a) = \bigvee_{s \in S_0} \bigvee_{p \in \rho(s,a)} p$
  - $\rho^a(s, a) = \bigvee_{p \in \rho(s,a)} p$
- $\overline{A}^a = \langle \Sigma, S' \times [2n], (s_0, 2n), \overline{\rho}^a, S' \times Odd([2n]) \rangle$, where
  - $\overline{\rho}^a((s_0, 2n), a) = \bigwedge_{s \in S_0} \bigwedge_{p \in \rho(s,a)} \bigvee_{i' \leq 2n} (p, i')$
  - $\overline{\rho}^a((s, i), a) = \begin{cases} \bigwedge_{p \in \rho(s,\pi)} \bigvee_{i' \leq i} (p, i') & s \notin F \text{ or } i \text{ is even} \\ false & s \in F \text{ and } i \text{ is odd} \end{cases}$

Claim 2.6.1 $L(A^a) = L(A)$

Proof: A run of A corresponds to a tree of A^a and vice versa. □

Claim 2.6.2 $L(\overline{A}^a) = \Sigma^\omega \setminus L(A)$

Proof: The proof is given in [KV97]. Considerable parts of the proof appear with variations also here. For an idea of the proof see Claim 2.6.4, Claim 3.2.4 and Subsection 3.5.1. □

2.6.2 Construction of the alternating automaton

Given an ETL_r formula g, we construct an alternating automaton $\mathcal{A}_g$ such that $L(\mathcal{A}_g) = L(g)$. We use the closure of the formula $cl(g)$ as the state set for this alternating automaton. Recall the definition of closure given for ETL_f and ETL_l formulas (Section 2.2). Recall also that for a formula g, the formula $\overline{g}$ denotes the negative normal form of $\neg g$ and the function $replace_A^S$ as defined in Section 2.3.

Theorem 2.6.3 *For every ETL_r formula g of length n there exists an alternating Büchi automaton $\mathcal{A}_g$ such that $L(\mathcal{A}_g) = L(g)$ and $\mathcal{A}_g$ has $O(n^2)$ states.*

Given an ETL_r formula g, we define $\mathcal{A}_g = \langle 2^{\mathcal{PROP}}, cl(g), g, \Delta, \mathcal{F}\rangle$, where the transition function Δ and the acceptance set $\mathcal{F}$ are defined as follows.

- The transition function $\Delta : cl(g) \times 2^{\mathcal{PROP}} \rightarrow B^+(cl(g))$ is defined as in previous sections. We recall part of the definition.
  - $\Delta(A_a(g_1, ..., g_n), \pi) = \bigvee_{i=1}^{n}[\Delta(g_i, \pi) \wedge replace_{A^a}^S(\rho_a^S(s_0, a_i))]$
    One of the formulas g_i is checked to hold and the computation of A^a continues. One run visits F infinitely often.
  - $\Delta(\overline{A}_a(g_1, ..., g_n), \pi) = \bigwedge_{i=1}^{n}[replace_{\overline{A}^a}^S(\overline{\rho}_a((s_0, 2n), a_i)) \vee \Delta(\overline{g_i}, \pi)]$
    Either g_i does not hold or the computation of $\overline{A}^a$ has to continue. No possible run visits F infinitely often.
- The acceptance set is

$$\mathcal{F} = \bigcup \begin{array}{l} \{A_s^a | A^a = \langle \Sigma, S, s_0, \rho^a, F^a\rangle \in cl(g) \text{ and } s \in F^a\} \\ \{\overline{A}_{(s,i)}^a | \overline{A}^a = \langle \Sigma, S \times [2n], (s_0, 2n), \overline{\rho}^a, S \times Odd([2n])\rangle \in cl(g) \text{ and i is odd}\} \end{array}$$

Unlike finite and looping acceptance automata for which it was sufficient to check only the positive or only the negative, here we have to check that both the negative and the positive automata visit infinitely often their acceptance sets.

Claim 2.6.4 $L(\mathcal{A}_g) = L(g)$

Proof: Prove by induction on the structure of the formula:

- Given the automaton connective $A(f_1, ..., f_n)$ where $A = \langle \Sigma, S, S_0, \rho, F\rangle$ with the alternating equivalent $A^a = \langle \Sigma, S', s_0, \rho^a, F^a\rangle$.

  A word $w = w_0 w_1 ...$ is a model for $A(f_1, ..., f_n)$ if there is an accepting run $\sigma = s_0, s_1, ...$ of A where for all $k \geq 0$ there is some $a_{j_k} \in \Sigma$ such that $(w, k) \models f_{j_k}$ and $s_{k+1} \in \rho(s_k, a_{j_k})$ and σ visits F infinitely often.

  By the induction assumption $(w, k) \models f_{j_k}$ if and only if $\mathcal{A}_{f_{j_k}}$ accepts the word $w_{\geq k}$ if and only if $\Delta(f_{j_k}, w_k)$ has an accepting run tree.

  We know that $A_{s_{k+1}}^a \models replace_{A^a}^S(\rho_a(s_k, a_{j_k}))$, so we can build the run tree of $\mathcal{A}_g$:
  - Label the root $A_{s_0}^a$

- Given a leaf x_k in the tree labeled by $A^a_{s_k}$, by the induction assumption there is an accepting run tree of $\mathcal{A}_{f_{j_k}}$ on $w_{\geq k}$ concatenate this tree under x_k (with x_k serving as the root) and add an extra leaf x_{k+1} labeled $A_{s_{k+1}}$
- Other leaves are parts of the subtree of $\mathcal{A}_{f_l}$ for some l. As we concatenated an accepting run of $\mathcal{A}_{f_l}$ we do not have to worry about these leaves. If a leaf appears in this subtree the transition associated with it has to be $\rho(V(x), a_{|x|}) = true$.

This is obviously a run of $\mathcal{A}_{A(f_1,...,f_n)}$. There is only one infinite path we have to worry about. This path is $A^a_{s_0}, A^a_{s_1}, ...$ which obviously visits the accepting set infinitely often.

A word $w = w_0 w_1 ...$ is in $L(\mathcal{A}_{A(f_1,...,f_n)})$ if there is an accepting run tree (T, V). There has to be in (T, V) an infinite path $x_0, x_1, ...$ labeled $A^a_{s_0}, A^a_{s_1}, ...$ The sequence $s_0, s_1, ...$ is an accepting run of $A(f_1, ..., f_n)$ on w.

- Given the automaton connective $\neg A(f_1, ..., f_n)$ where $A = \langle \Sigma, S, s_0, \rho, F \rangle$ with the alternating complement $\overline{A}^a = \langle \Sigma, S' \times [2n], (s_0, 2n), \overline{\rho}^a, S \times Odd([2n]) \rangle$.

  Suppose $w = w_0 w_1 ...$ does not satisfy $\neg A(f_1, ..., f_n)$. Then there exists an accepting run $\sigma = s_0, s_1, ...$ such that for all $k \geq 0$ there is $a_{j_k} \in \Sigma$ that $(w, k) \models f_{j_k}$ and $s_{k+1} \in \rho(s_k, a_{j_k})$ that visits F infinitely often.

  By contradiction suppose that (T, V) is an accepting run tree of $\mathcal{A}_{\neg A(f_1,...,f_n)}$ on w and build by induction a path that does not visit $\mathcal{F}$ infinitely often:

  - The root ϵ is labeled $\overline{A}^a_{(s_0,2n)}$
  - Given a path $\overline{A}^a_{(s_0,i_0)}, \overline{A}^a_{(s_1,i_1)}, ..., \overline{A}^a_{(s_m,i_m)}$

    We know that $\mathcal{A}_{f_{j_m}}$ accepts $w_{\geq m}$ hence there is no accepting tree for $\Delta(\overline{f_{j_m}}, w_m)$ so there is a node under $\overline{A}^a_{(s_m,i_m)}$ labeled $\overline{A}^a_{(s_{m+1},i_{m+1})}$

  We showed that $\overline{A}^a_{(s_0,i_0)}, \overline{A}^a_{(s_1,i_1)}, ...$ is a path in the tree (T, V). The sequence $i_0, i_1, ...$ is not increasing, therefore there exists some l such that for all $p \geq l, \quad i_p = i_l$. Since σ is an accepting run of $A(f_1, ..., f_n)$ it visits F infinitely often and there is no way that i_l is odd. The path we found in the tree visits $S \times Odd([2n])$ finitely often and the computation of $\mathcal{A}$ is rejecting. We have shown that $L(\mathcal{A}) \subseteq L(\neg A(f_1, ..., f_n))$ The other direction follows closely the proofs given in [KV97].

  Suppose $w \in L(\neg A(f_1, ..., f_n))$, there is no accepting run of $A(f_1, ..., f_n)$ on w. For all possible runs

  - either $\sigma = s_0, s_1, ...$ is infinite and for all $k \geq 0$ there is some $a_{j_k} \in \Sigma$ such that $(w, k) \models f_{j_k}$ and $s_{k+1} \in \rho(s_k, a_{j_k})$ and $lim(\sigma) \cap F = \emptyset$
  - or $\sigma = s_0, s_1, ..., s_m$ is finite and for all $0 \leq k < m$ there is some a_{j_k} such that $(w, k) \models f_{j_k}$ and $s_{k+1} \in \rho(s_k, a_{j_k})$ and $\forall a_l \in \Sigma, \quad (w, m) \not\models f_l$.

  We build the following labeled tree (T, V):

  - The root ϵ is labeled s_0
  - Given a leaf $x \in G$ labeled $V(x)$ define $SONS_x = \{s | \exists a_j \in \Sigma \text{ s.t. } s \in \rho(V(x), a_j)$ and $(w, |x|) \models f_j\}$. Let $|SONS_x| = m$ then add $x_1, ..., x_m$ as successors of x and label them with the values in $SONS_x$.

Given a tree run (T, V) we define the subtree of $x \in T$ as (T_x, V_x) where $T_x = \{y | x \cdot y \in T\}$ and $V_x(y) = V(x \cdot y)$. A tree (T, V) is defined *memoryless* if for every two nodes in the same level with the same label the subtrees below them are identical. Formally for all $x, y \in T$ such that $|x| = |y|$ and $V(x) = V(y)$, $(T_x, V_x) = (T_y, V_y)$. In a memoryless tree it seems a waste to hold more than a subset of S' per level. Since our tree is memoryless (see definition of $SONS_x$) we can convert it into a Directed Acyclic Graph $G = (V, E)$ where $V \subseteq S' \times \mathbb{N}$ and $E \subseteq \bigcup_{i=0}^{\infty}(S' \times \{i\}) \times (S' \times \{i+1\})$:
$V = \{(V(x), |x|) | x \in T\}$
$E = \{((V(x), |x|), (V(y), |y|)) | x, y \in T$ and y successor of x in $T\}$
From here on the proof is given in [KV97], we give here the main claims and the definitions used there.

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:

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

Lemma 2.6.5 [KV97] *For every $i \geq 0$, there exists l_i such that for all $l \geq l_i$, there are at most $n - i$ vertices of the form (s, i) in G_{2i}*

By the lemma G_{2n} is finite and hence G_{2n+1} 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 $[2n]$.

Lemma 2.6.6 [KV97] *For every two vertices (s, i) and (s', i') in G, if (s', i') is reachable from (s, i) then $rank(s', i') \leq rank(s, i)$.*

Lemma 2.6.7 [KV97] *In every infinite path in G, there exists a vertex (s, i) with an odd rank such that all the vertices (s', i') in the path that are reachable from (s, i) have $rank(s', i') = rank(s, i)$.*

We get back from [KV97] to the tree (T, V) and recall that the successors of x in T are $SONS_x = \{s | s \in \rho(V(x), a_j) \wedge (w, |x|) \models f_j\}$. We modify the tree:

- For the root ϵ we change the label to $\overline{A}^a_{(s_0, 2n)}$
- For every vertex $x \neq \epsilon$ we change the label to include its ranking: $\overline{A}^a_{(V(x), rank(V(x), |x|))}$. Since the rank is in the range $[2n]$, $\overline{A}^a_{(V(x), rank(V(x), |x|))}$ is indeed a state of the automaton.

Now for every x we append the following subtree. For all $a_j \in \Sigma$ such that $(w, |x|) \not\models f_j$ we add the computation of $\mathcal{A}_{\overline{f_j}}$ (with x as the root).

We first show that this is indeed a run of $\mathcal{A}_{\neg A(f_1,\ldots,f_n)}$, i.e. all nodes in the tree supply the transition function and that it is an accepting run:

- The root ϵ is labeled by $\overline{A}^a_{(s_0,2n)}$
  We divide the successors of ϵ to those labeled by $\overline{A}^a$ and those labeled by subformulas of $f_1, ..., f_n$. Since $2n$ is the maximal possible index all $\overline{A}^a$ are indexed below $2n$ and the transition is legal. The successors labeled by subformulas of $f_1, ..., f_n$ were added as a complete tree and obviously have legal transitions.
- For a node x labeled by $\overline{A}^a_{(s,i)}$
  We divide the successors of x to those labeled by $\overline{A}^a$ and those labeled by subformulas of $f_1, ..., f_n$. The node x and a successor labeled $\overline{A}^a$ are derived from two adjacent nodes in the tree (T, V). From Lemma 2.6.6 all successors of x have index smaller than i or equal to it. We also know that there is no way that $s \in F$ and i is odd. Again subformulas of $f_1, ..., f_n$ should not concern us.

The tree supplies the transition of the automaton.

By lemma 2.6.7, each infinite path of nodes labeled by $\overline{A}^a$ has a constant index from some level onward and that index is odd. The run is accepting.

□

2.6.3 From alternating Büchi automata to nondeterministic Büchi automata

Given an ETL$_r$ formula g we constructed the alternating automaton $\mathcal{A}_g$. In this section, given the alternating automaton $\mathcal{A}_g$, we use the construction in [MH84] and the methods discussed in the previous sections to transform it to a nondeterministic automaton. Let $|g| = n$, then $|cl(g)| = 2n^2$, implementing [MH84, Isl96] results in 3^{2n^2} states.

Theorem 2.6.8 *For every ETL$_r$ formula g of length n there exists a tight (loose) nondeterministic Büchi automaton B such that $L(B) = L(g)$ and B has $2^{O(n\ log(n))}$ states.*

Kupferman and Vardi [KV97] note that there is no point in using all the subsets of $cl(g)$. There is no need to hold a subset with the same state of an automaton with two different ranks. We can combine the tight and loose approaches with this observation to improve the construction.

In order to do so we extend the definition of a memoryless run to include two states with different indices.

Definition 1 *A* rank memoryless *run tree (T, V) of $\mathcal{A}_g$ is a memoryless run that has no two nodes in the same level labeled by $\overline{A}^a_{(s,i)}$ and $\overline{A}^a_{(s,i')}$ where $i \neq i'$ and $\overline{A}^a \in cl(g)$.*

Kupferman and Vardi [KV97] noted that a single run of a negative automaton results in a rank-memoryless run. In our case negative automata that are nested within other automata are spawned in different stages of the run. We cannot claim that the run is rank-memoryless and have to adapt it to be so.

Claim 2.6.9 *There is an accepting run of $\mathcal{A}_g$ on w iff there is a rank memoryless accepting run of $\mathcal{A}_g$ on w*

Proof: In a rank memoryless accepting run we have to replace every occurrence of $\overline{A}_{(s,i)}$ with no predecessor labeled by $\overline{A}$ by $\overline{A}_{(s,2n)}$. This does not affect the limit nature of the run and it remains an accepting run.

Given an accepting run (T,V) we transform it into a rank memoryless run by induction on the levels: For level 0 the tree is rank memoryless. Assume it is so until level i and show for $i+1$: Given two vertices in level $i+1$ labeled $\overline{A}^a_{(s,i)}$ and $\overline{A}^a_{(s,i')}$. W.l.o.g $i \leq i'$ and we replace the subtree under $\overline{A}^a_{(s,i')}$ (including $\overline{A}^a_{(s,i')}$) with the subtree under $\overline{A}^a_{(s,i)}$. The limit tree (T',V') is a rank memoryless tree and is still a valid run of $\mathcal{A}$. It is left to prove that (T',V') is accepting.

Assume by contradiction that (T',V') is not accepting, then there has to be an infinite path $x_0, x_1, ...$ in the tree that does not visit the acceptance set from some point onward. All the labels of this path are automata and if x_i is the parent of x_{i+1} in the tree either their labels belong to the same automaton or the automaton labeling x_{i+1} is nested within the automaton labeling x_i. Since the nesting degree is bounded, from some point in the path all labels are states of the same automaton. Hence either the path is labeled by a positive automaton that does not visit its accepting set or it is labeled by a negative automaton that is trapped in an even rank k. We show that either way a suffix of the path is included in the original tree.

In the first case, since there is a point from which the path is labeled by a positive automaton it cannot be the case that changes have been made to nodes in this path itself. Hence from this point onward the path is included in (T,V) and it has to be visiting F infinitely often.

In the second case all the labels belong from some point to the same negative automaton. We know as well that the ranks associated with this path are descending. The rank gets trapped in some k. Formally there exists some $i \geq 0$ such that for all $l \geq i$ the label of x_l is $\overline{A}^a_{(s_l,k)}$ for some automaton A and state $s_l \in S$. Show by induction that $\overline{A}_{(s_i,k)}, \overline{A}_{(s_{i+1},k)}, ...$ are the labels of a path in (T,V):

- Since $\overline{A}_{(s_i,k)}$ appears in (T',V') there is some node in level i in T with the same label. Let $y_0, y_1, ..., y_i$ be the path from the root to that node.
- Suppose $y_0, y_1, ..., y_m, \quad m \geq i$ is a path in (T,V) and the labels of $y_i, ..., y_m$ are labeled $\overline{A}_{(s_i,k)}, ..., \overline{A}_{(s_m,k)}$,
  Below y_m there is a node labeled $\overline{A}_{(s_{m+1},f)}$ for some $f \leq k$. But we know that in (T',V') appears in level $m+1$ a node with label $\overline{A}_{(s_{m+1},k)}$, hence k is the minimal rank appearing in (T,V) in level $m+1$ associated with s_{m+1}. We conclude that $f = k$.

Since (T,V) is an accepting run such a path cannot appear in it and we can conclude that (T',V') is also accepting. □

We combine the rank memoryless with the tight approach. We reduce the closure to contain one polarity of every formula $rcl(g)$ without ranks. We build a nondeterministic Büchi automaton with the consistent subsets of $\{\mp 2m, ..., \mp 0, -2m, ..., -0, 1, 2\}^{rcl(g)}$. Here m is the maximal number of states of all automata connectives $A(f_1, ..., f_n)$ nested in the formula g. Using this notation 1 indicates that the positive of the formula should be checked, $-i$ indicates that the negative of the

formula ranked i should be checked, 2 indicates that the positive of the formula should be checked and appears in the book-keeping component and $\mp i$ indicates that the negative of the formula ranked i should be checked and appears in the book-keeping component. A subset S is consistent if (a) boolean consistency of conjunctions and disjunctions is kept (b) a formula that is not an automaton connective always appears with rank 1 or -1.

We define the Büchi automaton $B = \langle 2^{\mathcal{PROP}}, S, S_0, \Delta', \alpha \rangle$ where S is the set of consistent subsets of $\{\mp 2m, ..., 2\}^{rcl(g)}$, S_0 contains all subsets in which g appears in the positive (rank 1) or $\overline{g}$ appears in the negative (rank $-2m, ..., -0$) and no state appears with ranks $2, \mp 2m, ..., \mp 0$. The acceptance set α includes all the sets in which no element is ranked $2, \mp 2m, ..., \mp 0$ (the book-keeping component is empty). The transition function requires that every positive automaton is followed by one possible successor and every negative automaton ranked i is followed by all possible successors ranked below i. We abuse notation and write (P, Q) as a state of B. For $f \in rcl(g)$ and A_s an automaton connective in $rcl(g)$ we abuse notation and write $f \in P$ meaning $P_f = 1$ (i.e. f's coordinate in P equals 1), $A_s \in P$ and $A_s \notin Q$ meaning $P_{A_s} = 1$, $A_s \in P$ and $A_s \in Q$ meaning $P_{A_s} = 2$ (Only automata might appear in Q), $(f, i) \notin P$ meaning $P_f = -i$ ($(A_s, i) \notin P$ and $\overline{A}_{(s,i)} \notin Q$ for automata connectives) and $(A_s, i) \notin P$ and $\overline{A}_{(s,i)} \in Q$ meaning $P_{A_s} = \mp i$.

The transition function Δ' is defined for all $(P, Q) \in S$ and $a \subseteq \mathcal{PROP}$ as follows.

- If $Q = \emptyset$, then $(P', Q') \in \Delta'((P, Q), a)$ iff all the following conditions hold.
  - For all $p \in \mathcal{PROP}$, we have $p \in P$ iff $p \in a$.
  - For all automata connectives $A(f_1, ..., f_n)$ with $A = \langle \Sigma, S, S_0, \rho, F \rangle$.
    - If $A_s \in P$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or for some i, $(\overline{f_j}, i) \notin P$ and $A_t \in P'$.
    - If $(A_s, i) \notin P$, then for all A_t such that $t \in \rho(s, a_j)$ where $f_j \in P$ or for some l, $(\overline{f_j}, l) \notin P$, there exists some $p \leq i$ and $(A_t, p) \notin P'$.
  - $Q' = \bigcup \begin{array}{l} \{A_s | A_s \in P' \text{ and } s \notin F\} \\ \{\overline{A}_{(t,p)} | (A_t, p) \notin P' \text{ and } p \text{ is even}\} \end{array}$
- If $Q \neq \emptyset$, then $(P', Q') \in \Delta'((P, Q), a)$ iff all the following conditions hold.
  - For all $p \in \mathcal{PROP}$, we have $p \in P$ iff $p \in a$.
  - For all automata connectives $A(f_1, ..., f_n)$ with $A = \langle \Sigma, S, S_0, \rho, F \rangle$.
    - If $A_s \in P$ and $A_s \notin Q$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or for some i, $(\overline{f_j}, i) \notin P$ and $A_t \in P'$.
    - If $(A_s, i) \notin P$ and $\overline{A}_{(s,i)} \notin Q$, then for all A_t such that $t \in \rho(s, a_j)$ where $f_j \in P$ or for some l, $(\overline{f_j}, l) \notin P$, there exists some $p \leq i$ and $(A_t, p) \notin P'$.
    - If $A_s \in P$ and $A_s \in Q$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or for some i, $(\overline{f_j}, i) \notin P$ and $A_t \in P'$ and either $t \in F$ or $A_t \in Q'$.
    - If $(A_s, i) \notin P$ and $\overline{A}_{(s,i)} \in Q$, then for all A_t such that $t \in \rho(s, a_j)$ where $f_j \in P$ or for some l, $(\overline{f_j}, l) \notin P$, there exists some $p \leq i$ and $(A_t, p) \notin P'$ and either p is odd or $\overline{A}_{(t,p)} \in Q'$.

Claim 2.6.10 $L(\mathcal{A}_g) = L(B)$

The proof is similar to the proof of claim 2.5.2. Proving that $L(B) \subseteq L(\mathcal{A}_g)$ we convert an accepting run of B to an accepting run tree of $\mathcal{A}_g$. Proving that $L(\mathcal{A}_g) \subseteq L(B)$ we build a Hintikka sequence for B and use the runs of $\mathcal{A}_g$ and $\mathcal{A}_f$ where f is a subformula of g to prove that the Hintikka sequence is indeed an accepting run of B.

We combine rank memoryless with the loose approach. The state set of the new automaton is the consistent sets in $\{\mp 2m, ..., \mp 0, -2m, ..., -0, 0, 1, 2\}^{rcl(g)}$. Again a state is consistent if (a) boolean consistency of disjunctions and conjunctions is kept and (b) subformulas that are not automata connectives appear with ranks $-0, 0$ and 1 only.

For $f \in rcl(g)$ and A_s an automaton connective in $rcl(g)$ we abuse notations and write $f \in P$ meaning $P_f = 1$ (i.e. f's coordinate in P equals 1), $A_s \in P$ and $A_s \notin Q$ meaning $P_{A_s} = 1$, $A_s \in P$ and $A_s \in Q$ meaning $P_{A_s} = 2$, $(f,i)\overline{\in}P$ meaning $P_f = -i$, $(A_s,i)\overline{\in}P$ and $\overline{A}_{(s,i)} \notin Q$ meaning $P_{A_s} = -i$, $(A_s,i)\overline{\in}P$ and $\overline{A}_{(s,i)} \in Q$ meaning $P_{A_s} = \mp i$ and $f \notin P$ meaning $P_f = 0$.

The nondeterministic Büchi automaton is $B = \langle 2^{\mathcal{PROP}}, S, S_0, \Delta', \alpha \rangle$ where S is the set of consistent sets in $\{\mp 2m, ..., \mp 0, -2m, ..., -0, 0, 1, 2\}^{rcl(g)}$, S_0 contains all the sets in which g appears with rank 1 or $\overline{g}$ appears with some negative rank and the book-keeping component is empty. The acceptance condition is all the states where the book-keeping component is empty.

The transition function Δ' is defined for all $(P,Q) \in S$ and $a \subseteq \mathcal{PROP}$ as follows.

- If $Q = \emptyset$, then $(P',Q') \in \Delta'((P,Q),a)$ iff all the following conditions hold.
  - For all $p \in \mathcal{PROP}$, we have $p \in P$ implies $p \in a$ and $p\overline{\in}P$ implies $p \notin a$.
  - For all automata connectives $A(f_1, ..., f_n)$ with $A = \langle \Sigma, S, S_0, \rho, F \rangle$.
    - If $A_s \in P$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or for some i, $(\overline{f_j}, i)\overline{\in}P$ and $A_t \in P'$.
    - If $(A_s,i)\overline{\in}P$, then for all A_t such that $t \in \rho(s,a_j)$ where $f_j \in P$ or for some l, $(\overline{f_j}, l)\overline{\in}P$ or ($f_j \notin P$ and $\overline{f_j} \notin P$), there exists some $p \leq i$ and $(A_t,p)\overline{\in}P'$.
  - $Q' = \bigcup \begin{array}{l} \{A_s | A_s \in P' \text{ and } s \notin F\} \\ \{\overline{A}_{(t,p)} | (A_t,p)\overline{\in}P' \text{ and } p \text{ is even}\} \end{array}$
- If $Q \neq \emptyset$, then $(P',Q') \in \Delta'((P,Q),a)$ iff all the following conditions hold.
  - For all $p \in \mathcal{PROP}$, we have $p \in P$ implies $p \in a$ and $p\overline{\in}P$ implies $p \notin a$.
  - For all automata connectives $A(f_1, ..., f_n)$ with $A = \langle \Sigma, S, S_0, \rho, F \rangle$.
    - If $A_s \in P$ and $A_s \notin Q$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or for some i, $(\overline{f_j}, i)\overline{\in}P$ and $A_t \in P'$.
    - If $(A_s,i)\overline{\in}P$ and $\overline{A}_{(s,i)} \notin Q$, then for all A_t such that $t \in \rho(s,a_j)$ where $f_j \in P$ or for some l, $(\overline{f_j}, l)\overline{\in}P$ or ($f_j \notin P$ and $\overline{f_j} \notin P$), there exists some $p \leq i$ and $(A_t,p)\overline{\in}P'$.
    - If $A_s \in P$ and $A_s \in Q$, then there exists some $a_j \in \Sigma$ and $t \in \rho(s, a_j)$ such that $f_j \in P$ or for some i, $(\overline{f_j}, i)\overline{\in}P$ and $A_t \in P'$ and either $t \in F$ or $A_t \in Q'$.
    - If $(A_s,i)\overline{\in}P$ and $\overline{A}_{(s,i)} \in Q$, then for all A_t such that $t \in \rho(s,a_j)$ where $f_j \in P$ or for some l, $(\overline{f_j}, l)\overline{\in}P$ or ($f_j \notin P$ and $\overline{f_j} \notin P$), there exists some $p \leq i$ and $(A_t,p)\overline{\in}P'$ and either p is odd or $\overline{A}_{(t,p)} \in Q'$.

Claim 2.6.11 $L(\mathcal{A}_g) = L(B)$

The proof is similar to previous proofs and is omitted.

Chapter 3

Extending temporal logic with alternating automata

As suggested in the last section of [VW94], we extend temporal logic with alternating automata, we call this logic ETL_a. Since alternating automata are as expressive as nondeterministic automata, the expressive power of ETL_a is equal to that of ETL_r[1]. Although alternating automata are exponentially more succinct the translation from ETL_a to nondeterministic Büchi automata has the same complexity.

3.1 Definition of ETL_a

Syntax Formulas are defined with respect to a set $Prop$ of propositions.

- Every proposition $p \in Prop$ is a formula.
- If f_1 and f_2 are formulas , then $\neg f_1$, $f_1 \vee f_2$ and $f_1 \wedge f_2$ are formulas.
- For every alternating finite automaton $A = \langle \Sigma, S, \rho, s_0, F \rangle$ with $\Sigma = \{a_1, ..., a_n\}$. If $f_1, ..., f_n$ are formulas, then $A(f_1, ..., f_n)$ is a formula.

Semantics The satisfaction of a formula is defined with respect to a model $\pi \in (2^{\mathcal{PROP}})^\omega$ and a location $i \in \mathbb{N}$. Given an infinite word $\pi \in (2^{\mathcal{PROP}})^\omega$ and a location $i \in \mathbb{N}$ we define satisfaction:

- For a proposition $p \in \mathcal{PROP}$, we have $(\pi, i) \models p$ iff $p \in \pi_i$.
- $(\pi, i) \models \neg f_1$ iff not $(\pi, i) \models f_1$.
- $(\pi, i) \models f_1 \vee f_2$ iff $(\pi, i) \models f_1$ or $(\pi, i) \models f_2$
- $(\pi, i) \models f_1 \wedge f_2$ iff $(\pi, i) \models f_1$ and $(\pi, i) \models f_2$

Consider an automaton $A = \langle \Sigma, S, s_0, \rho, F \rangle$. The *run* of the formula $A(f_1, ..., f_n)$ over a word π starting at point i, is a finite or infinite S-labeled tree (T, V) such that $V(\epsilon) = s_0$ and for

[1] The expressiveness power of ETL_f, ETL_l and ETL_r are all equal [VW94]

all nodes $x \in T$ there is some $a_j \in \Sigma$ such that $(\pi, i + |x|) \models f_j$ and the (possibly empty) set $P = \{V(y) | y \text{ is a successor of } x \text{ in } T\}$ satisfies the transition $\rho(V(x), a_j)$.

A run is *accepting* if every infinite path of T visits F infinitely often. We can now complete the definition of semantics:

- $(\pi, i) \models A(f_1, ..., f_n)$ iff there is an accepting run of $A(f_1, ..., f_n)$ over π starting at i.

3.2 Translating ETL_a formulas into nondeterministic Büchi automata

As in the case nondeterministic automata, given an ETL_a formula we build a nondeterministic Büchi automaton that accepts the same language. Again, this is done in two stages, first construct an alternating Büchi automaton and then convert the alternating Büchi automaton into a nondeterministic Büchi automaton.

3.2.1 Complementing an alternating automaton

We create for every automaton connective an alternating automaton accepting the same language and an alternating automaton accepting the complementary language. The one accepting the same language is already given to us. We build an automaton accepting the complementary language.

Given an alternating automaton $A = \langle \Sigma, S, s_0, \rho, F \rangle$ the dual automaton is a co-Büchi automaton accepting the complementary language. Kupferman and Vardi [KV97] build a weak alternating automaton that accepts the complementary language (a weak alternating automaton is both Büchi and co-Büchi). We use the same notation used in previous chapter.

$$\overline{A}^a = \langle \Sigma, S \times [2n], (s_0, 2n), \overline{\rho}^a, S' \times Odd([2n]) \rangle$$

In order to define the transition function we follow the notation used in [KV97]. We define the function $release : B^+(S) \times [2n] \rightarrow B^+(S \times [2n])$. Given a formula $\phi \in B^+(S)$, and a rank $i \in [2n]$, the formula $release(\phi, i)$ is obtained from ϕ by replacing an atom $s \in S$ by the disjunction $\bigvee_{i' \leq i}(s, i')$. Recall the definition of ρ^d, the dual of ρ (obtained from ρ by replacing $\wedge$ with $\vee$ and vice versa).

$$\overline{\rho}^a((s, i), \pi) = \begin{cases} release(\rho'(s, \pi), i) & s \notin F \text{ or } i \text{ is even} \\ false & s \in F \text{ and } i \text{ is odd} \end{cases}$$

Claim 3.2.1 $L(\overline{A}) = \Sigma^\omega \setminus L(A)$

Proof: The proof is given in [KV97] □

3.2.2 Construction of the alternating Büchi automaton

Given an ETL_a formula g, we construct an alternating automaton $\mathcal{A}_g$ such that $L(\mathcal{A}_g) = L(g)$.

Theorem 3.2.2 *For every ETL_a formula g of length n there exists an alternating Büchi automaton $\mathcal{A}_g$ such that $L(\mathcal{A}_g) = L(g)$ and $\mathcal{A}_g$ has $O(n^2)$ states.*

For the state set of this alternating automaton we use the closure of the formula $cl(g)$. Recall the definition of closure from previous sections. Again all formulas in the closure are assumed to be in negative normal form. Recall also that the function $replace_A^S$ takes a formula ϕ and replaces an element s by A_s.

Given an ETL_a formula g, we define $\mathcal{A}_g = \langle 2^{\mathcal{PROP}}, cl(g), g, \Delta, \mathcal{F} \rangle$, where the transition function Δ and the acceptance set $\mathcal{F}$ are defined as follows.

- The transition function $\Delta : cl(g) \times 2^{\mathcal{PROP}} \rightarrow B^+(cl(g))$ is defined as in previous sections. We recall part of the definition.
  - $\Delta(A(g_1, ..., g_n), a) = \bigvee_{i=1}^{n} [\Delta(g_i, a) \wedge replace_A^S(\rho(s_0, a_i))]$
  - $\Delta(\overline{A}(g_1, ..., g_n), a) = \bigwedge_{i=1}^{n} [replace_{\overline{A}}^S(\overline{\rho}((s_0, 2n), a_i)) \vee \Delta(\overline{g_i}, a)]$
- The acceptance set is

$$\mathcal{F} = \bigcup \begin{array}{l} \{A_s | A = \langle \Sigma, S, s_0, \rho, F \rangle \in cl(g) \text{ and } s \in F\} \\ \{\overline{A}_{(s,i)} | \overline{A} = \langle \Sigma, S \times [2n], (s_0, 2n), \overline{\rho}, S \times Odd([2n]) \rangle \in cl(g) \text{ and } i \text{ is odd}\} \end{array}$$

Claim 3.2.3 $L(\mathcal{A}_g) = L(g)$

Proof: The proof is very similar to the proof in the case of ETL_r. The fact that a memoryless run exists was proven in [EJ91]. □

3.2.3 From alternating Büchi automata to nondeterministic Büchi automata

As in the previous chapters, we can use [MH84] to convert the alternating automaton into a nondeterministic automaton. Given a formula g with $|g| = n$, the size of $\mathcal{A}_g$ is $O(n^2)$ and we get a nondeterministic automaton with $2^{O(n^2)}$ states.

Again this can be reduced to $2^{O(n\ log(n))}$. In order to use the methods of the previous section we have to show a rank-memoryless run. Recall Definition 1. No negative automaton can appear in the same state with two different ranks. In the previous chapter the deterministic nature of the run of negative automata was used in the proof.

An *alternating parity automaton* is a tuple $P = \langle \Sigma, Q, q_0, \rho, \alpha \rangle$ where Σ, Q, q_0 and ρ are like before and $\alpha = \{F_0, ..., F_m\}$ is a subset of 2^Q . A run of a parity automaton on a word w is defined like the run of an alternating Büchi automaton. The run is accepting if for every infinite path in the run of P there is an even i such that the path visits F_i infinitely often and visits $F_{i'}$ for $i' < i$ only finitely often.

Claim 3.2.4 *There is an accepting run of $\mathcal{A}_g$ on w iff there is a rank memoryless accepting run of $\mathcal{A}_g$ on w.*

Proof: Once again a rank memoryless run is a run. In order to show the other direction we would like to combine Büchi and co-Büchi conditions in one automaton. We use alternating parity automata.

Given an ETL_a formula g we built an alternating Büchi automaton $\mathcal{A}_g$ by incorporating into it alternating automata for positive automata formulas and alternating automata for negative automata formulas. Using the parity acceptance condition we can avoid the complementation construction for alternating automata as following. Given a negative automaton connective $\neg A(f_1, ..., f_n)$ we can build A^d the dual of A, a co-Büchi automaton.

Now for every positive automaton connective we have an equivalent alternating Büchi automaton and for every negative automaton connective we have an equivalent co-Büchi automaton. We plug the co-Büchi automaton into the alternating parity automaton instead of the complementary automaton we have built. We define the acceptance set $\alpha = \{F_1, F_2, F_3\}$ of the parity automaton as follows.

For every positive automaton $A = \langle \Sigma, Q, q_0, \rho, F \rangle$ we unite with F_2 the set $\{A_s | s \in F\}$ and with F_3 the set $\{A_s | s \notin F\}$. This way if F is visited infinitely often F_2 will be visited infinitely often otherwise F_3 will be visited infinitely often and F_2 only finitely often. For every negative automaton (co-Büchi) $A^d = \langle \Sigma, Q, q_0, \rho^d, F \rangle$, we unite with F_1 the set $\{A^d_s | s \in F\}$ and with F_2 the set $\{A^d_s | s \notin F\}$. This way if F is visited infinitely often F_1 will be visited infinitely often, otherwise F_2 will be visited infinitely often. Or more formally

$$F_1 = \{A^d_s | A(f_1, ..., f_n) \in cl(g),\ A^d = \langle \Sigma, S, s_0, \rho^d, F \rangle \text{ and } s \in F\}$$

$$F_2 = \bigcup \begin{array}{l} \{A^d_s | A(f_1, ..., f_n) \in cl(g),\ A^d = \langle \Sigma, S, s_0, \rho^d, F \rangle \text{ and } s \notin F\} \\ \{A_s | A(f_1, ..., f_n) \in cl(g),\ A = \langle \Sigma, S, s_0, \rho, F \rangle \text{ and } s \in F\} \end{array}$$

$$F_3 = \{A_s | A(f_1, ..., f_n) \in cl(g),\ A = \langle \Sigma, S, s_0, \rho, F \rangle \text{ and } s \notin F\}$$

The formula g defines a natural partial order on the elements in $cl(g)$. Enhance this order into a well order $g_1 < g_2 < ... < g_l = g$. We are interested only in automata connectives because propositions and boolean disjunctions and conjunctions do not appear as labels in the run tree of $\mathcal{A}_g$ (except maybe as the label of the root).

Take a node in the run tree of the parity automaton $\mathcal{A}_g$. If this node is labeled by some formula $f \in cl(g)$ then the successors of x will be labeled by formulas that are before f in the above order. That is $V(x \cdot c) \leq V(x)$. As every descending chain is finite in the order we know that every infinite path eventually gets trapped, i.e. for every path π there exists some node $x \in \pi$ such that for all $y \in \mathbb{N}^*$ such that $x \cdot y \in \pi$, we have $V(x) \leq V(x \cdot y)$ and $V(x \cdot y) \leq V(x)$. As only automata and propositions label the nodes in the run tree of $\mathcal{A}_g$, this means that the label of x and all its descendents on the path π are labeled by some automaton connective with different states. This fact ensures that the acceptance condition of the parity automaton is sound.

Emerson and Jutla [EJ91] have shown that alternating parity automata have memoryless runs. So we can restrict our attention to memoryless runs of the parity automaton. Now we convert it into a Büchi automaton by applying the ranking method of Kupferman and Vardi [KV97]. Every negative automaton is augmented with a ranking just like we had in the first place.

We show how to replace an automaton connective appearing in F_1 and F_2 by a connective appearing in F_2 and F_3. After F_1 is left empty we have a parity automaton $\mathcal{A} = \langle 2^{\mathcal{PROP}}, S, s_0, \Delta, \alpha \rangle$ with $\alpha = \{F'_2, F'_3\}$. The language of this automaton is equal to the language of the Büchi automaton $\mathcal{A}' = \langle 2^{\mathcal{PROP}}, S, s_0, \Delta, F'_2 \rangle$. Take an accepting run of $\mathcal{A}$, every infinite path visits F'_2 infinitely often, hence it is also an accepting run of $\mathcal{A}'$. Take an accepting run of $\mathcal{A}'$, every infinite path visits F'_2 infinitely often, hence it is also an accepting run of $\mathcal{A}$, the minimal set in α a path visits is even.

Given a memoryless accepting run tree (T, V) of $\mathcal{A}_g$ on a word w we convert it to a DAG run $G = (V, E)$ where $V = \{(V(x), |x|) | x \in T\}$ and $E = \{((V(x), |x|), (V(y), |y|)) | x, y \in T$ and y successor of x in $T\}$.

We are only interested in the co-Büchi automaton $A^d = \langle \Sigma, Q, q_0, \rho^d, F \rangle$. Given a DAG $G' \subseteq G$, we change the definitions accordingly. A vertex (s, i) is *eventually safe* in G' iff only finitely many vertices labeled by states of A^d (i.e. for some state $q \in Q$, A^d_q) are reachable in G' from (s, i). A vertex (s, i) is *currently safe* in G' iff all the vertices labeled by states of A^d reachable in G' from (s, i) are not members of $F \times \mathbb{N}$. Notice that automata nested within $A(f_1, ..., f_n)$ in the formula g are eventually safe. Indeed, no vertex labeled by A is reachable from them.

The inductive sequence:

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

Lemma 3.2.1 [KV97] *For every $i \geq 0$, there exists l_i such that for all $l \geq l_i$, there are at most $n - i$ vertices of the form (A^d_s, i) in G_{2i}.*

As we changed the definition of currently safe and eventually safe, the same proof from [KV97] works also here.

We give the vertices labeled by states of S ranks as in [KV97]. Rank (A^d_s, l) by $2i$ if the vertex is eventually safe in G_{2i}. Rank (A^d_s, l) by $2i + 1$ if the vertex is currently safe in G_{2i+1}.

Lemma 2.6.6 and Lemma 2.6.7 apply also here. In the definition of $\mathcal{A}_g$ we replace the states of type A^d_s by $A^d_{(s,i)}$ where $i \in [2n]$. We replace $A^d_{q_0}$ by $A^d_{(q_0,2n)}$ and modify the transition of $\mathcal{A}_g$ for the new states:

$$\Delta(A^d_{(s,i)}, a) = \begin{cases} release(\Delta(A^d_s, a), i) & s \notin F \text{ or } i \text{ is even} \\ false & s \in F \text{ and } i \text{ is odd} \end{cases}$$

We remove from F_1 and F_2 all the states of A^d, add to F_2 the states $A^d_{(s,i)}$ where i is odd, and add to F_3 the states $A_{(s,i)}$ where i is even.

As we started from a memoryless run of the parity automaton and handled all the states of the negative automata together, we conclude that the run of the resulting Büchi automaton is rank memoryless. □

Theorem 3.2.5 *For every ETL_a formula g of length n there exists a tight (loose) nondeterministic Büchi automaton B such that $L(B) = L(g)$ and B has $2^{O(n\ log(n))}$ states.*

Given the alternating automaton $\mathcal{A}_g$ the construction of the nondeterministic automaton B is very similar to the construction described in the previous chapter.

3.3 Extending temporal logic with 2-way alternating automata

The final stage is to enhance the logic with 2-way alternating automata, we call this logic ETL_{2a}. Once again given an ETL_{2a} formula g we build a nondeterministic Büchi automaton that accepts

exactly the models of the formula. The stages are similar to the work in previous chapters and proceeds as follows. We complement a 2-way alternating Büchi automaton. We build a 2-way alternating Büchi automaton that accepts the models of g. We then show that we cannot convert a 2-way alternating automaton into a 1-way alternating automaton avoiding an exponential blowup. Consequently, we use a larger alphabet as extra memory. We define a 1-way alternating automaton over a larger alphabet and a projection from the larger alphabet on the original alphabet. The projection of the language of the 1-way alternating automaton is the language of the 2-way alternating automaton. Our final step is translating the alternating automaton into a nondeterministic automaton.

3.4 Definition of ETL_{2a}

Syntax Formulas are defined with respect to a set $Prop$ of propositions.

- Every proposition $p \in Prop$ is a formula.
- If f_1 and f_2 are formulas , then $\neg f_1$, $f_1 \vee f_2$ and $f_1 \wedge f_2$ are formulas.
- For every 2-way alternating finite automaton $A = \langle \Sigma, S, \rho, s_0, F \rangle$ with $\Sigma = \{a_1, ..., a_n\}$. If $f_1, ..., f_n$ are formulas, then $A(f_1, ..., f_n)$ is a formula.

Semantics The satisfaction of a formula is defined with respect to a model $\pi \in (2^{\mathcal{PROP}})^\omega$ and a location $i \in \mathbb{N}$. Given an infinite word $\pi \in (2^{\mathcal{PROP}})^\omega$ and a location $i \in \mathbb{N}$ we define satisfaction:

- For a proposition $p \in \mathcal{PROP}$, we have $(\pi, i) \models p$ iff $p \in \pi_i$.
- $(\pi, i) \models \neg f_1$ iff not $(\pi, i) \models f_1$.
- $(\pi, i) \models f_1 \vee f_2$ iff $(\pi, i) \models f_1$ or $(\pi, i) \models f_2$.
- $(\pi, i) \models f_1 \wedge f_2$ iff $(\pi, i) \models f_1$ and $(\pi, i) \models f_2$.

Consider an automaton $A = \langle \Sigma, S, s_0, \rho, F \rangle$. The *run* of the formula $A(f_1, ..., f_n)$ over a word π starting at point i, is a finite or infinite $(S \times \mathbb{N})$-labeled tree (T, V) such that $V(\epsilon) = (s_0, i)$ and for all $x \in T$, let $V(x) = (s, k)$, then there is some $a_j \in \Sigma$ such that $(\pi, k) \models f_j$ and the (possibly empty) set $P = \{(s', c') | y \text{ successor of } x \text{ in } T \text{ and } V(y) = (s', k + c')\}$ satisfies the transition $\rho(V(x), a_j)$.

The run is accepting if every infinite path of T visits $F \times \mathbb{N}$ infinitely often. We can complete the definition of semantics:

- $(\pi, i) \models A(f_1, ..., f_n)$ iff there is an accepting run of $A(f_1, ..., f_n)$ over π starting at i.

3.5 Translating ETL_{2a} formulas into 2-way alternating Büchi automata

Similar to the previous sections, given a ETL_{2a} formula g, we construct a 2-way alternating automaton $\mathcal{A}_g$ such that $L(\mathcal{A}_g) = L(g)$. Our first step is to given a 2-way alternating Büchi automaton, construct a 2-way alternating Büchi automaton accepting the complementary language.

3.5.1 Complementing a 2-way alternating automaton

We claim that the construction of Kupferman and Vardi [KV97] works also here. Once again we repeat the main claims and definitions. This time we have to prove some of the claims. Let $A = \langle \Sigma, Q, q_0, \rho, F \rangle$ be a 2-way alternating Büchi automaton. Its dual $A^d = \langle \Sigma, Q, q_0, \rho^d, F \rangle$ is a 2-way alternating co-Büchi automaton accepting the complementary language. We analyze the run of A^d in order to construct a 2-way alternating Büchi automaton that accepts the same language as A^d (the complement of A).

Theorem 2 *If a 2-way co-Büchi automaton A' accepts a word w, then there exists a memoryless accepting run of A' on w.*

Proof: Emerson and Jutla [EJ91] showed that if a 1-way co-Büchi automaton accepts a word w, then there exists a memoryless accepting run of the automaton on w. Their proof consists of building a ranking function that depends only on the future of the run. The same proof works also for 2-way runs. □

Given a 2-way alternating co-Büchi automaton $A = \langle \Sigma, Q, q_0, \rho, F \rangle$ and an accepting run (T, V) of A on a word w, we can represent the run using a directed (probably cyclic) graph G where $V = \{V(x) | x \in T\}$ and $E = \{(V(x), V(y)) | x, y \in T \text{ and } y \text{ successor of } x \text{ in } T\}$. Given a node $x \in T$ with label $V(x) = (s, i)$, the node x relates to letter i of the input word. As the automaton is a 2-way automaton the successors of x may relate to letter i again, go backwards to read $i - 1$ or go forward to read letter $i + 1$. Thus, V is still a subset of $Q \times \mathbb{N}$ but E is a subset of

$$\bigcup_{i=0}^{\infty} \left(\bigcup \begin{array}{l} ((Q \times \{i\}) \times (Q \times \{i+1\})) \\ ((Q \times \{i\}) \times (Q \times \{i\})) \\ ((Q \times \{i+1\}) \times (Q \times \{i\})) \end{array} \right)$$

Once again given a (possibly finite) directed graph $G' \subseteq G$. 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:

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

Lemma 3.5.1 [KV97] *For every $i \geq 0$, there exists l_i such that for all $l \geq l_i$, there are at most $n - i$ vertices of the form (s, i) in G_{2i}*

Proof: We follow the proof in [KV97]. The induction base case is immediate. Assume the lemma's requirement holds for i. Consider G_{2i}, in the case it is finite, then G_{2i+1} is empty, G_{2i+2} is empty as well, and we are done. Otherwise, there must exist some currently safe vertex in G_{2i+1}. Assume by contradiction that G_{2i} is infinite and no vertex in G_{2i+1} is currently safe. Since G_{2i} is infinite so is G_{2i+1} and every vertex in G_{2i+1} has at least one successor. Consider some vertex (q_0, l_0) in

G_{2i+1}. By the assumption it is not currently safe, so there is some vertex (q_1, l_1) reachable from (q_0, l_0) where $q_1 \in F$ is a member of the set F. Let (q_2, l_2) be a successor of (q_1, l_1), by assumption (q_2, l_2) is also not currently safe. We can continue and build by induction an infinite path in G_{2i+1} that visits F infinitely often. But this path is also a path in (T, V) contradicting the assumption that (T, V) is an accepting run.

We diverge here from the proof in [KV97]. Let (q, l) be a currently safe vertex in G_{2i+1}. We show that removing it and all its descendants results in a thinner graph. Denote the subgraph of all the vertices reachable from (q, l) by G'. Since (q, l) is in G_{2i+1}, G' is infinite and all nodes in G' are currently safe. We define an ordering on the nodes in G' according to (a) the minimal distance from the vertex (q, l) (b) the level in the graph (c) some ordering on Q. Obviously this is a well order. Let G'' be a subgraph of G'. G'' contains all vertices in G' but every vertex has at most one predecessor, the minimal predecessor in G' according to the ordering. The graph G'' is a tree.

There are no cycles in G''. A cycle cannot include (q, l), since it has no predecessors. Suppose (q', l') is the minimal node in a cycle. The shortest path from (q, l) to (q', l') in G' cannot pass through one of the nodes in the cycle. Hence (q', l') does not choose any of the nodes in the cycle as its predecessor.

The graph G'' remains connected. Assume otherwise, there is a connected component that is not reachable from (q, l). Take the minimal node in that connected component (q', l'). There is a shortest path in G', connecting (q, l) to (q', l'). The predecessor of (q', l') along this path cannot be in the connected component of (q', l'), contradiction.

We have shown that G'' is an infinite tree. By König's lemma this tree contains an infinite *diverging* path α, this path does not return to the same vertex twice. Define $l_{i+1} = max(l, l_i)$, we know that for every $k > l_{i+1}$ the path α visits level k in the graph G_{2i+1}. All nodes on α are not eventually safe in G_{2i} and are currently safe in G_{2i+1} hence they are not in G_{2i+2}. We are done. □

By the lemma, G_{2n} is finite and hence G_{2n+1} is empty.

Lemma 3.5.1 [KV97] *For every two vertices (s, i) and (s', i') in G, if (s', i') is reachable from (s, i) then $rank(s', i') \leq rank(s, i)$.*

Lemma 3.5.2 [KV97] *In every infinite path in G, there exists a vertex (s, i) with an odd rank such that all the vertices (s', i') in the path that are reachable from (s, i) have $rank(s', i') = rank(s, i)$.*

The proof of the above two Lemmas follows [KV97].

Given a 2-way alternating Büchi automaton $A = \langle \Sigma, Q, q_0, \rho, F \rangle$ the complement automaton is $\overline{A} = \langle \Sigma, Q \times [2n], (q_0, 2n), \overline{\rho}, Q \times Odd([2n]) \rangle$ where $\overline{\rho} = release(\rho^d)$ and ρ^d is the dual of ρ.

3.5.2 Construction of the 2-way alternating Büchi automaton

We construct now a 2-way alternating Büchi automaton that accepts the set of models of an ETL_{2a} formula g. The method is similar to the constructions in previous sections.

Theorem 3.5.3 *For every ETL_{2a} formula g of length n there exists an 2-way alternating Büchi automaton $\mathcal{A}_g$ such that $L(\mathcal{A}_g) = L(g)$ and $\mathcal{A}_g$ has $O(n^2)$ states.*

As before, the state set of the 2-way alternating automaton is the closure of the formula g. Recall that the closure consists of formulas in negative normal form. We use the function $replace_A^S$ defined in previous chapters.

Given the ETL_{2a} formula g, we define $\mathcal{A}_g = \langle 2^{\mathcal{PROP}}, cl(g), g, \Delta, \mathcal{F} \rangle$, where the transition function Δ and the acceptance set $\mathcal{F}$ are defined as follows.

- The transition function $\Delta : cl(g) \times 2^{\mathcal{PROP}} \to B^+(\{-1, 0, 1\} \times cl(g))$ is defined by induction.
  - $\Delta(p, a) = \begin{cases} true & p \in a \\ false & p \notin a \end{cases}$
  - $\Delta(\neg p, a) = \begin{cases} true & p \notin a \\ false & p \in a \end{cases}$
  - $\Delta(g_1 \wedge g_2, a) = (g_1, 0) \wedge (g_2, 0)$
  - $\Delta(g_1 \vee g_2, a) = (g_1, 0) \vee (g_2, 0)$
  - $\Delta(A(g_1, ..., g_n), a) = \bigvee_{i=1}^{n} [(g_i, 0) \wedge replace_A^S(\rho(s_0, a_i))]$
  - $\Delta(\overline{A}(g_1, ..., g_n), a) = \bigwedge_{i=1}^{n} [replace_{\overline{A}}^S(\overline{\rho}((s_0, 2n), a_i)) \vee (\overline{g_i}, 0)]$
- The acceptance set is

$$\mathcal{F} = \bigcup \begin{array}{l} \{A_s | A = \langle \Sigma, S, s_0, \rho, F \rangle \in cl(g) \text{ and } s \in F\} \\ \{\overline{A}_{(s,i)} | \overline{A} = \langle \Sigma, S \times [2n], (s_0, 2n), \overline{\rho}, S \times Odd([2n]) \rangle \in cl(g) \text{ and } i \text{ is odd}\} \end{array}$$

Note that here, unlike previous sections, instead of defining Δ recursively we spawn states that read the same letter and check the correctness of a sub expression.

Claim 3.5.4 $L(\mathcal{A}_g) = L(g)$

Proof: We prove by induction on the structure of the formula g. Note that it is not enough to simply walk on the parse tree of the formula g. This time if a process is spawned when the automaton is reading w_i it may go backwards to read the letters of w occuring before i. The induction assumption is that if the automaton spawns a copy in state s reading letter w_i this copy accepts iff $(w, i) \models s$. The proof for propositions and boolean connectives is immediate.

- For a formula $g = A(g_1, ..., g_n)$ where $A = \langle \Sigma, S, s_0, \rho, F \rangle$.

  If $(w, k) \models A(g_1, ..., g_n)$ we know there exists an accepting run tree of A on w. The labels of the tree T are from the set $S \times \mathbb{N}$. We convert this tree run into a tree run of $\mathcal{A}_g$ starting at letter w_k. Assuming that $\mathcal{A}_g$ is spawned reading letter w_k, it may go further back until w_0. We also add the runs of $\mathcal{A}_{g_i}$. Note that this time we add the runs of $\mathcal{A}_{g_i}$ as is, we take the root of the run and add it **under** the node in the tree run of $\mathcal{A}_g$ (and not unite the roots like in previous cases).

  The other direction is similar. Given an accepting tree run of $\mathcal{A}_g$ starting from letter w_k, from the induction assumption an accepting run of $\mathcal{A}_{g_i}$ starting from letter w_j exists iff $(w, j) \models g_i$. We can prune the tree to serve as a run tree of the formula g on the word w starting from k.

- For a formula $G = \neg A(g_1, ..., g_n)$ where $A = \langle \Sigma, S, s_0, \rho, F \rangle$ and $\overline{A} = \langle \Sigma, S \times [2n], (s_0, 2n), \overline{\rho}, S \times Odd([2n]) \rangle$.

  Suppose $(w, k) \models A(g_1, ..., g_n)$. The accepting runs of $A(g_1, ..., g_n)$ on w starting at k and $\mathcal{A}_{\neg A(g_1,...,g_n)}$ on w starting at k cannot co-exist. There are two paths $s_0, s_1, ...$ and $(s_0, 2n), (s_1, i_1), ...$ the first in the run of $A(g_1, ..., g_n)$ and the second in the run of $\mathcal{A}_{\neg A(g_1,...,g_n)}$. The first should visit F infinitely often and thus the second cannot be trapped in a set $S \times \{2i + 1\}$. Hence $L(\mathcal{A}_{\neg A(g_1,...,g_n)}) \subseteq L(A(g_1, ..., g_n))$.

  Given that there is no accepting run of $A(g_1, ..., g_n)$ on w starting at k. We build by induction a tree of states from $S \times \mathbb{N}$. We start with (s_0, k). We assume by induction that for a leaf (s, l) the formula $A_s(g_1, ..., g_n)$ does not hold on w (starting at l). Obviously for (s_0, k) the assumption holds.

  For a leaf x labeled (s, l), since $(w, l) \not\models A_s(g_1, ..., g_n)$ for every letter $a_j \in \Sigma$ such that $(w, l) \not\models \overline{f_j}$ (that is $(w, l) \models f_j$) there exists a set of states and directions $\{(s_1, c_1), ..., (s_2, ...c_2)\}$ that satisfies the dual of the transition $\rho(s, a_j)$, $l + c_i \geq 0$ and $(w, l + c_i) \not\models A_{s_i}(g_1, ..., g_n)$ for all is. If such a set does not exist then $(w, l) \models A_s(g_1, ..., g_n)$ contrary to the assumption. So for every letter a_j such that $(w, l) \not\models \overline{f_j}$ we add this set of successors to the leaf x.

  According to [EJ91] we can also find such a memoryless tree. We build the Directed Graph just like in Section 3.5.1 and show that the we can rank the states with the set $[2n]$.

  Our last step is to complete the tree T with ranks and with the subtrees of the computations of $\overline{g_j}$ when appropriate. The resulting tree is a valid run of $\mathcal{A}$ starting at k. The infinite behavior of the tree supplies the acceptance condition $S \times Odd([2n])$.

□

This completes the construction of $\mathcal{A}_g$. We review the options of converting 2-way automata to 1-way automata.

3.6 Transforming 2-way automata to 1-way automata

We would like now to convert the 2-way alternating Büchi automaton into a 1-way alternating Büchi automaton. In order to give a uniform treatment to the different extended temporal logics, we would like to continue working with alternating automata. As we show in Appendix A, given a 2-way nondeterministic Büchi automaton we can construct a 1-way alternating Büchi automaton recognizing the same language. The number of states of the alternating automaton is polynomial in the number of states of the nondeterministic one.

We would have liked to do a similar construction for 2-way alternating Büchi automata. Thus, given a 2-way alternating Büchi automaton, we want to construct an equivalent 1-way alternating Büchi automaton of polynomial size. A lower bound by Birget [Bir93] claims that there is an exponential gap between 2-way and 1-way alternating finite automata. We enhance this lower bound to apply also for Büchi automata.

We would like to avoid an exponential blowup when transforming a 2-way alternating automaton to a 1-way alternating automaton. In order to do so we use the alphabet as extra memory. Given a 2-way automaton we construct a 1-way automaton over a larger alphabet and a homomorphism between the two alphabets (see [HU87]). Birjet [Bir96] has shown that the language of a 2-way

alternating finite automaton is a homomorphic image of the language of a polynomial size 1-way alternating finite automaton. We enhance this result to alternating Büchi automata.

3.6.1 A lower bound on the conversion of 2-way alternating automata to 1-way alternating automata

Birjet [Bir93] showed that the best conversion from 2-way alternating finite automata to one-way alternating automata is exponential. We enhance this result to alternating Büchi automata on words.

Theorem 3 [Bir93] *For every n there exists a language $L \subseteq \Sigma^*$ and a 2-way alternating finite automaton with n states accepting L such that the minimal 1-way alternating finite automaton accepting L has at least 2^{n-2} states.*

Assume that $\sharp \notin \Sigma$. We prove the following two claims:

Claim 3.6.1 *Given a two-way alternating finite automaton with n states accepting the language L, one can construct a two-way alternating Büchi automaton accepting the language $L \cdot \sharp^\omega$ with $O(n)$ states.*

Claim 3.6.2 *Given a 1-way alternating Büchi automaton with n states accepting the language $L \cdot \sharp^\omega$, one can construct a 1-way alternating finite automaton accepting L with $O(n)$ states.*

From the two claims the following corollary follows:

Corollary 4 *For every n, there exists a language $L \subseteq \Sigma^\omega$ and a 2-way alternating Büchi automaton with n states accepting L such that a 1-way alternating Büchi automaton accepting L has at least $2^{\Omega(n)}$ states.*

Proof: [Claim 3.6.1] Given a 2-way alternating finite automaton $U = \langle \Sigma, Q, q_0, \rho, F \rangle$ accepting $L \subseteq \Sigma^*$ we construct $U' = \langle \Sigma \cup \{\sharp\}, Q \cup \{q_\sharp\}, q_0, \rho', \{q_\sharp\} \rangle$ accepting the language $L \cdot \sharp^\omega$. The transition function is defined:

$$\rho'(q,a) = \begin{cases} \rho(q,a) & \text{if } q \in Q \text{ and } a \neq \sharp \\ (q_\sharp, 1) & \text{if } (q \in F \text{ or } q = q_\sharp) \text{ and } a = \sharp \\ false & \text{if } q \notin F \text{ and } q \neq q_\sharp \text{ and } a = \sharp \\ false & \text{if } q = q_\sharp \text{ and } a \neq \sharp \end{cases}$$

We show now that U' accepts exactly $L \cdot \sharp^\omega$. Given a finite word $w \in L$ there exists an accepting run of U on w. Since the run is accepting it is a finite run and all its leaves are states from F. We append an infinite path under each one of these vertices labeled by $(q_\sharp, |w| + i)$. The new infinite tree is an accepting run of U'.

Given an accepting run of U' on a word $w \in (L \cup \{\sharp\})^\omega$ we show that $w = w' \cdot \sharp^\omega$ for some $w' \in L$. Take the accepting run of U', since the run is accepting every path visits $q_\sharp$ infinitely often. Since $q_\sharp$ is a sink reading only $\sharp$ signs the word is of the form $\Sigma^* \cdot \sharp^\omega$. Furthermore since $q_\sharp$ moves only forward we can prune all the infinite paths labeled by $q_\sharp$ and get an accepting run of U. $\square$

Proof: [Claim 3.6.2] Given an 1-way alternating Büchi automaton $A = \langle \Sigma \cup \{\sharp\}, Q, q_0, \rho, F\rangle$ accepting $L \cdot \sharp^\omega$ we construct $A' = \langle \Sigma, Q, q_0, \rho', F'\rangle$ where ρ' is the restriction of ρ to Σ and $F' = \{q | A^q \text{ accepts } \sharp^\omega\}$ (where A^q is the automaton A with start state q).

An accepting run tree of A' on a word w can be easily converted into an accepting run tree of A on $w \cdot \sharp^\omega$. An accepting run of A on a word $w \cdot \sharp^\omega$ can be pruned into an accepting run tree of A'. All the states appearing in level $|w|$ in the tree have to appear in F', they accept the suffix $\sharp^\omega$. □

3.7 From 2-way alternating Büchi automata to 1-way alternating Büchi automata

Given a 2-way alternating finite automaton $A = \langle \Sigma, S, s_0, \rho, F\rangle$, Birjet [Bir96] has shown that there exists an alphabet Σ', a function $p : \Sigma' \to \Sigma$ and a 2-way alternating finite automaton $A' = \langle \Sigma', S', s'_0, \rho', F'\rangle$ such that if we enhance p to words in Σ'^* and to subsets of Σ'^* in the natural way, $p(L(A')) = L(A)$. The number of states of A' need not be more than polynomial in the number of states of A.

We prove a similar result for 2-way alternating Büchi automata. Given a 2-way alternating Büchi automaton $A = \langle \Sigma, Q, q_0, \delta, F\rangle$, we give two alphabets. The alphabet of A, namely Σ and another alphabet Δ_A that depends on the structure of A. We build a 1-way alternating automaton $B = \langle \Sigma \times \Delta_A, Q', q'_0, \delta', F'\rangle$ whose alphabet is $\Sigma \times \Delta_A$. As an homomorphism we use the projection on the first component. More formally, we define the projection $p_1 : \Sigma \times \Delta_A \to \Sigma$, as $p(a, b) = a$. (enhanced to $(\Sigma \times \Delta_A)^\omega$ and subsets of $(\Sigma \times \Delta_A)^\omega$ in the natural way) such that $p_1(L(B)) = L(A)$. In particular, A recognizes the empty language iff B recognizes the empty language.

3.7.1 The construction

The details follow Vardi [Var98]. As Vardi solved the problem of converting 2-way alternating parity tree automata into 1-way nondeterministic parity tree automata we have to modify slightly his work. To each letter of the alphabet we add a strategy, a way to satisfy the transition of A, and an annotation, a finite representation of backward runs.

Let $A = \langle \Sigma, Q, \delta, q_0, F\rangle$ be a 2-way alternating Büchi automaton.

Definition 5 *A* strategy *for A is a mapping $\tau : \mathbb{N} \to 2^{Q \times \{-1,0,1\} \times Q}$. For each label $\zeta \subseteq Q \times \{-1, 0, 1\} \times Q$, define state$(\zeta) = \{u : (u, i, u') \in \zeta\}$. The strategy τ is* on *a word w if $q_0 \in$ state$(\tau(0))$, and for all $i \in \mathbb{N}$ and each state $q \in$ state$(\tau(i))$, the set $\{(c, q') | (q, c, q') \in \tau(i)\}$ satisfies $\delta(q, w_i)$.*

A *path* in a strategy τ is a finite or infinite sequence $(0, q_0), (i_1, q_1), (i_2, q_2), ...$ of pairs from $\mathbb{N} \times Q$ such that, either the path is infinite and for all $j \geq 0$, there is some $c_j \in \{-1, 0, 1\}$ such that $(q_j, c_j, q_{j+1}) \in \tau(i_j)$ and $i_{j+1} = i_j + c_j$, or the path is finite $(0, q_0), ..., (i_m, q_m)$ and for all $0 \leq j < m$, there is some $c_j \in \{-1, 0, 1\}$ such that $(q_j, c_j, q_{j+1}) \in \tau(i_j)$, $i_{j+1} = i_j + c_j$ and $\delta(q_m, w_{i_m}) =$ '*true*'. A path is defined accepting if it visits $\mathbb{N} \times F$ infinitely often or if it finite. We say that τ is *accepting* if all infinite paths in τ are accepting.

Proposition 3.7.1 [Var98] *A two-way alternating Büchi automaton accepts a word iff it has an accepting strategy on the word.*

An *annotation* for A is a mapping $\eta : \mathbb{N} \rightarrow 2^{Q \times \{\bot, \top\} \times Q}$. In the following discussion we regard $\bot < \top$ as an ordering on the pair. We also use

$$\chi_F(q) = \begin{cases} \top & \text{if } q \in F \\ \bot & \text{if } q \notin F \end{cases}$$

the characteristic function of F.

We say that η is an annotation *of* the strategy τ (which in turn is on the word w) if the following closure conditions hold for all $i \in \mathbb{N}$.

1. if $(q, \alpha, q') \in \eta(i)$ and $(q', \beta, q'') \in \eta(i)$ then $(q, max(\alpha, \beta), q'') \in \eta(i)$.
2. if $(q, 0, q') \in \tau(i)$ then $(q, \chi_F(q'), q') \in \eta(i)$.
3. if $i > 0$, $(q, -1, q') \in \tau(i)$, $(q', \alpha, q'') \in \eta(i-1)$ and $(q'', 1, q''') \in \tau(i-1)$ then $(q, max(\chi_F(q'), \alpha, \chi_F(q''')), q''') \in \eta(i)$.
4. if $(q, 1, q') \in \tau(i)$, $(q', \alpha, q'') \in \eta(i+1)$ and $(q'', -1, q''') \in \tau(i+1)$ then $(q, max(\chi_F(q'), \alpha, \chi_F(q''')), q''') \in \eta(i)$.
5. if $i > 0$, $(q, -1, q') \in \tau(i)$ and $(q', 1, q'') \in \tau(i-1)$ then $(q, max(\chi_F(q'), \chi_F(q'')), q'') \in \eta(i)$.
6. if $(q, 1, q') \in \tau(i)$ and $(q', -1, q'') \in \tau(i+1)$ then $(q, max(\chi_F(q'), \chi_F(q'')), q'') \in \eta(i)$.

A *downward path* k in η is a sequence $(i_1, q_1, t_1), (i_2, q_2, t_2), ...$ of triplets, where each i_j is in $\mathbb{N}$, each q_j is in Q, each t_j is either an element of $\tau(i_j)$ or $\eta(i_j)$, and

- Either t_j is $(q_j, 1, q_{j+1})$ and $i_{j+1} = i_j + 1$ in this case we record a visit to the accepting set if $q_{j+1} \in F$.
- Or t_j is (q_j, α, q_{j+1}) where $\alpha \in \{\bot, \top\}$ and $i_{j+1} = i_j$ in this case we record a visit to the accepting set if $\alpha = \top$.

A downward path can be finite, if the last triplet is (i_m, q_m, t_m) and $t_m = (q, \alpha, q)$ (i.e. the path ends in a loop) or the last triplet is (i_m, q_m, t_m) and $\delta(q_m, w_{i_m}) = true$. A finite path is accepting in two cases. Let (i_m, q_m, t_m) be the last triplet in the path, then it is accepting if either $\delta(q_m, w_{i_m}) = true$ or $t_m = (q, \top, q)$. An infinite path is accepting if it visits the acceptance set infinitely often.

Proposition 3.7.2 [Var98] *A two-way alternating Büchi automaton accepts a word iff it has a strategy on the word and an accepting annotation of the strategy.*

Given a 2-way alternating Büchi automaton $A = \langle \Sigma, Q, q_0, \delta, F \rangle$ we define two alphabets $\Delta^s_Q \subseteq 2^{Q \times \{-1,0,1\} \times Q}$, $\Delta^a_Q = 2^{Q \times \{\bot, \top\} \times Q}$. Denote $\Sigma' = \Sigma \times \Delta^s_Q \times \Delta^a_Q$. We define three projections $p_1 : \Sigma' \rightarrow \Sigma$, $p_2 : \Sigma' \rightarrow \Delta^s_Q$ and $p_3 : \Sigma' \rightarrow \Delta^a_Q$, the projections on the first, second and third components. We enhance the projections in the natural way for infinite words and sets of infinite words.

We build the 1-way alternating Büchi automaton $B = \langle \Sigma', Q', q'_0, \rho, F' \rangle$ such that $p_1(L(B)) = L(A)$. We build B in two stages. First B_1 makes sure that $p_2(w)$ is a strategy on $p_1(w)$ and that

$p_3(w)$ is an annotation of the strategy $p_2(w)$. Second we build B_2 that checks that all downward paths visit F infinitely often.

We start with B_1. Most of the conditions B_1 has to check are local conditions. In order to check the conditions of the strategy and the first two conditions of the annotation we can check each entry $(a, \zeta, \mu) \in \Sigma'$. So we restrict Σ' to include only the letters that supply these *local conditions*. Consecution of the strategy and conditions 3-6 of the annotation involve relation between two letters and are checked by B_1.

Let $B_1 = \langle \Sigma', Q_1, q_0^1, \rho_1, \{q_1^1\} \rangle$ where

$$Q_1 = \bigcup \begin{array}{l} \{q_0^1, q_1^1\} \\ \{c\} \times Q \times \{\in, \notin\} \\ \{s\} \times Q \times \{-1\} \times Q \times \{\notin\} \\ \{a\} \times Q \times \{\bot, \top\} \times Q \times \{\in, \notin\} \end{array}$$

. The four kinds of states are:

- The states q_0^1 and q_1^1, when reading letter (a, ζ, μ) spawn all processes that check consecution of the strategy and conditions 3-6 of the annotation. The state q_0^1 checks that q_0 is in the state set of the second element of the current letter. Both spawn q_1^1 to check recursively the rest of the word.
- The states labeled by c check consecution of the strategy. If there is some state $(q, 1, q')$ in the current strategy there should be a strategy for q' in the next strategy. If there is no strategy for q' in the current strategy the next strategy should not contain states of the form $(q, -1, q')$.
- The states labeled by a represent a triple (q, α, q') of the annotation that should belong $(\in)$ or not belong $(\notin)$ to the third element of the current letter.
- The states labeled by s represent a triple $(q, -1, q')$ of the strategy that should not belong to the second element of the current letter.

The transition of B_1 is defined as following:

- $\rho((c, q, \in), (a, \zeta, \mu)) = \begin{cases} true & \text{if } q \in state(\zeta) \text{ or } \delta(q, a) = \text{`}true' \\ false & \text{Otherwise} \end{cases}$
- $\rho((c, q, \notin), (a, \zeta, \mu)) = \begin{cases} false & \text{if there exists } q' \text{ s.t. } (q', -1, q) \in \zeta \\ true & otherwise \end{cases}$
- $\rho((a, q_1, \alpha, q_2, \in), (a, \zeta, \mu)) = \begin{cases} true & \text{if } (q_1, \alpha, q_2) \in \mu \\ false & \text{if } (q_1, \alpha, q_2) \notin \mu \end{cases}$
- $\rho((a, q_1, \alpha, q_2, \notin), (a, \zeta, \mu)) = \begin{cases} false & \text{if } (q_1, \alpha, q_2) \in \mu \\ true & \text{if } (q_1, \alpha, q_2) \notin \mu \end{cases}$
- $\rho((s, q_1, i, q_2, \notin), (a, \zeta, \mu)) = \begin{cases} false & \text{if } (q_1, i, q_2) \in \zeta \\ true & \text{if } (q_1, i, q_2) \notin \zeta \end{cases}$

- Define $consec(a,\zeta) = \{q \in Q | q \notin state(\zeta) \text{ and } \delta(q,a) \neq true\}$.
  Let $\phi^c(a,\zeta,\mu) = \bigwedge_{q \in consec(a,\zeta)} (c,q,\notin) \wedge \bigwedge_{(q',1,q)\in\zeta} (c,q,\in)$
  This represents the consecution of the strategy.

- Define $\phi^3(a,\zeta,\mu) = \bigwedge_{(q,1,q')\in\zeta} \bigwedge_{(q,\alpha,q''')\notin\mu} \bigwedge_{q''\in Q} [(a,q',\alpha',q'',\notin) \vee (s,q'',-1,q''',\notin)]$
  where $\alpha = max(\alpha', \chi_F(q'), \chi_F(q''))$. This represents condition 3 of the annotation.

- Define $\phi^4(a,\zeta,\mu) = \bigwedge_{(q',\alpha,q'')\in\mu} \bigwedge_{(q'',1,q''')\in\zeta} \bigwedge_{q\in Q} [(s,q,-1,q',\notin) \vee (a,q,\alpha',q''',\in)]$
  where $\alpha' = max(\alpha, \chi_F(q'), \chi_F(q'''))$. This represents condition 4 of the annotation.

- Define $\phi^5(a,\zeta,\mu) = \bigwedge_{(q,1,q')\in\zeta} \bigwedge_{(q,\alpha,q'')\notin\mu} (s,q,-1,q'',\notin)$
  Where $\alpha = max(\chi_F(q'), \chi_F(q''))$. This represents condition 5 of the annotation.

- Define $\phi^6(a,\zeta,\mu) = \bigwedge_{(q',1,q'')\in\zeta} \bigwedge_{q\in Q} [(s,q,-1,q',\notin) \vee (a,q,\alpha,q'',\in)]$
  Where $\alpha = max(\chi_F(q'), \chi_F(q''))$. This represents condition 6 of the annotation.

- $\rho(q_0^1,(a,\zeta,\mu)) = \begin{cases} q_1^1 \wedge \phi^c(a,\zeta) \wedge \phi^3(a,\zeta,\mu) \wedge \phi^4(a,\zeta,\mu) \wedge \phi^5(a,\zeta,\mu) \wedge \phi^6(a,\zeta,\mu) & \text{if } q_0 \in state(\zeta) \\ false & \text{Otherwise} \end{cases}$

- $\rho(q_1^1,(a,\zeta,\mu)) = q_1^1 \wedge \phi^c(a,\zeta) \wedge \phi^3(a,\zeta,\mu) \wedge \phi^4(a,\zeta,\mu) \wedge \phi^5(a,\zeta,\mu) \wedge \phi^6(a,\zeta,\mu)$

The consecution of the strategy checks that either the transition of a state is '$true$' or the computation has to continue. The correctness of the annotation conditions results from the following simple logical equivalences:

$$\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \rightarrow \varphi_4 \equiv \varphi_1 \wedge (\neg\varphi_4) \rightarrow (\neg\varphi_2) \vee (\neg\varphi_3)$$

$$\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \rightarrow \varphi_4 \equiv \varphi_1 \wedge \varphi_2 \rightarrow \varphi_4 \vee (\neg\varphi_3)$$

The state q_0^1 ensures that q_0 has some strategy. Since q_1^1 is always spawned the strategy is on the word and the annotation is of the strategy (other conditions are part of the alphabet).

We turn now to the second automaton. We define $B_2 = \langle \Sigma', Q \times \{\bot,\top\}, \rho, (q_0,\bot), \rho_2, F \times \{\bot\} \cup Q \times \{\top\} \rangle$, where the transition function ρ_2 is defined as follows.

$$\rho_2((q,\alpha),(a,\zeta,\mu)) = \begin{cases} false & \text{if } \begin{array}{l}(q,\bot,q) \in \mu \text{ or} \\ \exists q' \text{s.t. } (q,\alpha,q') \in \mu \text{ and } (q',\bot,q') \in \mu\end{array} \\ \bigwedge_{(q,1,q')\in\zeta}(q',\bot) \wedge \bigwedge_{(q,\beta,q')\in\mu} \bigwedge_{(q',1,q'')\in\zeta} (q'',\beta) & \text{Otherwise} \end{cases}$$

There is one difference from the definition of a downward path. A downward path could take 0-steps, i.e. read a triple from the annotation and stay reading the same letter. From the closure condition of the annotation we see that if $(q,\alpha,q') \in \mu$ and $(q',\beta,q'') \in \mu$ then so is $(q, max(\alpha,\beta), q'') \in \mu$. If a downward path makes a finite sequence of steps reading the same letter in w, $(q_0,\alpha_0,q_1),...,(q_m,\alpha_m,q_{m+1})$ and then taking a forward step $(q_{m+1},1,q_{m+2})$ from the

strategy, we can model this behavior by $(q_0, max(\alpha_i), q_{m+1}) \in \mu$ and $(q_{m+1}, 1, q_{m+2}) \in \zeta$. If a downward path makes an infinite sequence of steps reading the same letter in w there has to be some state q' appearing infinitely often in the sequence and by the same closure property there exists $(q, \alpha, q') \in \mu$ and $(q'\beta, q') \in \mu$.

We take now B_1 and B_2 and combine them to a single automaton B that is the conjunction of the two.

3.7.2 From alternating Büchi automata to nondeterministic Büchi automata

Recall that given a formula g of length n the 2-way alternating automaton $\mathcal{A}_g = \langle 2^{\mathcal{PROP}}, cl(g), g, \Delta, \mathcal{F} \rangle$ has $O(n^2)$ states. Hence, the 1-way alternating automaton in this case has $O(n^4)$ states. If we use [MH84] to convert this alternating automaton to a nondeterministic Büchi automaton, we get a nondeterministic automaton with $2^{O(n^4)}$ states. We endeavor to reduce this to $2^{O(n^2 log(n))}$.

Theorem 3.7.1 *For every ETL_{2a} formula g of length n there exists a nondeterministic Büchi automaton B such that $L(B) = L(g)$ and B has $2^{O(n^2 log(n))}$ states.*

We define the consistent subsets of $cl(g) \times \{-1, 0, 1\} \times cl(g)$ (strategy) and of $cl(g) \times \{\bot, \top\} \times cl(g)$ (annotation). Intuitively in a consistent subset there cannot appear a formula and its negation and also a negative formula with more than one rank. We show that a rank memoryless run exists also here. The strategy of such a run is consistent and the minimal annotation of that strategy is also consistent.

We start with the strategy. For a subset $\zeta \subseteq cl(g) \times \{-1, 0, 1\} \times cl(g)$, $state(\zeta) = \{q | (q, c, q') \in \zeta\}$ and $target(\zeta) = \{(c, q') | (q, c, q') \in \zeta\}$. A subset ζ is consistent if for every automaton connective $A(g_1, ..., g_n)$:

1. If $A_s \in state(\zeta)$ then for all ranks i, $\overline{A}_{(s,i)} \notin state(\zeta)$
2. If $\overline{A}_{(s,i)} \in state(\zeta)$ then $A_s \notin state(\zeta)$.
3. If $\overline{A}_{(s,i)} \in state(\zeta)$ then for all other ranks $i' \neq i$, $\overline{A}_{(s,i')} \notin state(\zeta)$.
4. If $(c, A_s) \in target(\zeta)$ then for all ranks i, $(c, \overline{A}_{(s,i)}) \notin target(\zeta)$
5. If $(c, \overline{A}_{(s,i)}) \in target(\zeta)$ then $(c, A_s) \notin target(\zeta)$
6. If $(c, \overline{A}_{(s,i)}) \in target(\zeta)$ then for all other ranks $i' \neq i$, $(c, \overline{A}_{(s,i')}) \notin target(\zeta)$

Denote $CONS$ the set of consistent subsets of $cl(g) \times \{-1, 0, 1\} \times cl(g)$.

Definition 6 *A* rank memoryless strategy *for A is a mapping $\tau : \mathbb{N} \rightarrow CONS$. The rest of the definition is similar to Definition 5.*

The annotation is rank memoryless if it supplies similar conditions. For a subset $\mu \subseteq cl(g) \times \{\bot, \top\} \times cl(g)$ define $current(\mu) = \{s | \exists s', \alpha \text{ s.t. } (s, \alpha, s') \in \mu \text{ or } (s', \alpha, s) \in \mu\}$. Consider the letter of Σ', (a, ζ, μ)

1. If $A_s \in current(\mu)$ then for all ranks i, $\overline{A}_{(s,i)} \notin current(\mu)$

2. If $\overline{A}_{(s,i)} \in current(\mu)$ then for no other rank $i' \neq i$, $\overline{A}_{(s,i')} \in current(\mu)$

3. If $\overline{A}_{(s,i)} \in current(\mu)$ then $A_s \notin current(\mu)$

4. If $A_s \in current(\mu)$ then $A_s \in state(\zeta)$

5. If $\overline{A}_{(s,i)} \in current(\mu)$ then $\overline{A}_{(s,i)} \in state(\zeta)$

We now show that $\mathcal{A}_g$ has a rank memoryless run and that the strategy of this run and the annotation of that strategy are also rank memoryless.

Claim 3.7.2 *A two-way alternating Büchi automaton accepts an input w iff it has an accepting rank memoryless run on w.*

Proof: One direction is simple. A rank memoryless run is a run. We combine the proofs of Claim 3.2.4 and Lemma 3.5.1. We build an alternating parity automaton and show using the ordering (1) distance from the root (2) level in the graph (3) some order on the state set how to get a rank memoryless tree from the run of the parity automaton. □

The strategy applied by the automaton in this rank memoryless run is a rank memoryless strategy. This strategy also supplies another condition. Given three consecutive letters $(a_i, \zeta_i, \mu_i), (a_{i+1}, \zeta_{i+1}, \mu_{i+1}), (a_{i+2}, \zeta_{i+2}, \mu_{i+2})$ we know that if $(1, \overline{A}_{(s,i)}) \in target(\zeta_i)$ and $(-1, \overline{A}_{(s,i')}) \in target(\zeta_{i+2})$ then $i = i'$ (and $\overline{A}_{(s,i)} \in state(\zeta_{i+1})$).

Given two annotations of the same strategy η_1 and η_2 we know that their intersection $\eta_1 \cap \eta_2$ defined by $\eta_1 \cap \eta_2(x) = \eta_1(x) \cap \eta_2(x)$ is also an annotation of the strategy. So if we take the minimal annotation of the rank memoryless strategy it is rank memoryless by the following claim.

Claim 3.7.3 *If a triple (q, α, q') appears in the minimal annotation of the letter w_i with the strategy of some run, there are states q and q' that read letter w_i in that run.*

Proof: We prove the claim by induction on the closure properties of the annotation.

1. For the condition: if $(q, \alpha, q') \in \eta(i)$ and $(q', \beta, q'') \in \eta(i)$ then $(q, max(\alpha, \beta), q'') \in \eta(i)$. By induction in the run of $\mathcal{A}_g$ there are state q and q'' reading letter w_i.

2. For the condition: if $(q, 0, q') \in \tau(i)$ then $(q, \chi_F(q'), q') \in \eta(i)$. The rank memoryless strategy was obtained from the rank memoryless run, hence there is a state q reading w_i and it has a successor q' reading also w_i.

3. For the condition: if $i > 0$, $(q, -1, q') \in \tau(i)$, $(q', \alpha, q'') \in \eta(i-1)$ and $(q'', 1, q''') \in \tau(i-1)$ then
$(q, max(\chi_F(q'), \alpha, \chi_F(q''')), q''') \in \eta(i)$. The state q reading letter w_i from the strategy. The state q'' reading w_{i-1} from the induction assumption and then by the strategy q''' reads w_i.

4. For the condition: if $(q, 1, q') \in \tau(i)$, $(q', \alpha, q'') \in \eta(i+1)$ and $(q'', -1, q''') \in \tau(i+1)$ then
$(q, max(\chi_F(q'), \alpha, \chi_F(q''')), q''') \in \eta(i)$. Similar to condition number 3.

5. For the condition: if $i > 0$, $(q, -1, q') \in \tau(i)$ and $(q', 1, q'') \in \tau(i-1)$ then $(q, max(\chi_F(q'), \chi_F(q'')), q'') \in \eta(i)$. States q and q'' reading letter w_i from the strategy.

6. For the condition: if $(q, 1, q') \in \tau(i)$ and $(q', -1, q'') \in \tau(i{+}1)$ then $(q, max(\chi_F(q'), \chi_F(q'')), q'') \in \eta(i)$. States q and q'' reading letter w_i from the strategy.

□

Since the run is rank memoryless so is the annotation.

We further restrict the alphabet Σ' to adhere to these new rules about the strategy and the annotation.

Recall that given an alternating automaton $B = \langle \Sigma, Q, q_0, \rho, F \rangle$ we get a nondeterministic automaton $N = \langle \Sigma, 2^Q \times 2^Q, \rho', 2^Q \times \emptyset \rangle$ where a state $(P, Q) \in 2^Q \times 2^Q$ always conforms to $Q \subseteq P$.

The state set of $\mathcal{A}_g$, denoted D, is the union of:

$$\{q_0^1, q_1^1\}$$

$$\{c\} \times cl(g) \times \{\in, \notin\}$$

$$\{a\} \times cl(g) \times \{\bot, \top\} \times cl(g) \times \{\in, \notin\}$$

$$\{s\} \times cl(g) \times \{-1\} \times cl(g) \times \{\in, \notin\}$$

$$cl(g) \times \{\bot, \top\}$$

In order to understand which subsets of D will suffice, we analyze the transition. Take the sequence $(a_1, \zeta_1, \mu_1), (a_2, \zeta_2, \mu_2)$. The current letter read by the automaton is (a_1, ζ_1, μ_1). We have to decide on the subset of states spawned to read (a_2, ζ_2, μ_2). We have to show that holding one rank per automaton in this subset is enough.

Making the states labeled by c (consecution of the strategy) memoryless is easy. We spawn a state $(c, \overline{A}_s, \notin)$ (with no rank) only if this negative automaton does not appear at all in the current strategy, and we spawn a state $(c, \overline{A}_{(s,i)}, \notin)$ if exactly this rank of the automaton $\overline{A}_s$ appears in the current strategy. In this case $(c, \overline{A}_{(s,i)}, \notin)$ checks that only rank i of the automaton $\overline{A}_s$ may appear in the strategy of the next letter. The states $(c, \overline{A}_{(s,i)}, \in)$ will be spawned with one rank only from the consistency of the alphabet.

In order to show a similar result for the states labeled by s and by a we give different semantics to states $(s, q, -1, q', \notin)$ when they are part of ϕ^3 and ϕ^5 and when they are part of ϕ^4 and ϕ^6. Hence we double the number of states by considering $\{s_{(3,5)}, s_{(4,6)}\} \times cl(g) \times \{-1\} \times cl(g) \times \{\notin\}$. Dealing with ϕ^3 and ϕ^5 is a bit more complex than ϕ^4 and ϕ^6.

We start with ϕ^4.

$$\phi^4(a, \zeta, \mu) = \bigwedge_{(q', \alpha, q'') \in \mu} \bigwedge_{(q'', 1, q''') \in \zeta} \bigwedge_{q \in Q} [(s, q, -1, q', \notin) \vee (a, q, \alpha', q''', \in)]$$

The states q' and q''' appear in $current(\mu_1)$ and $target(\zeta_1)$. Therefore if either of the two is a negative automaton, it may appear with only one rank. So all states of the form $(s_{(4,6)}, q, -1, \overline{A}_{(s,j)}, \notin)$ have $j = i$, similarly for $(a, q, \alpha', \overline{A}_{(s,j)}, \notin)$. The state q appears in a conjunction over all possible states. If we take a negative automaton $\overline{A}$ in state r, either there exists some i such that $\overline{A}_{(r,i)}$ belongs to $state(\zeta_2)$ or for all is $\overline{A}_{(r,i)}$ does not belong to $state(\zeta_2)$. If the second is correct then even if we do not give rank to $(s_{(4,6)}, \overline{A}_r, -1, q', \notin)$ the state will hold over ζ_2. If the first is correct

then there is some rank i for which $(s_{(4,6)}, \overline{A}_r, -1, q', \notin)$ does not hold. We require that for the same rank $(a, \overline{A}_r, \alpha', q''', \in)$ will hold. But from the consistency of ζ_2 and μ_2 the only possible rank of $\overline{A}_r$ appearing in μ_2 is i. We conclude that there is no need to rank the state q, we interpret $(s_{(4,6)}, q, -1, q', \notin)$ as true only if q does not appear at all in ζ_2. We interpret $(a, q, \alpha', q''', \in)$ as true if some rank of q appears in μ_2.

The analysis of ϕ^6 is similar.

$$\phi^6(a, \zeta, \mu) = \bigwedge_{(q',1,q'') \in \zeta} \bigwedge_{q \in Q} [(s, q, -1, q', \notin) \vee (a, q, \alpha, q'', \in)]$$

Again q appears in all possible ranks and we interpret $(s_{(4,6)}, q, -1, q', \notin)$ as true if q does not appear at all in ζ_2.

$$\phi^3(a, \zeta, \mu) = \bigwedge_{(q,1,q') \in \zeta} \bigwedge_{(q,\alpha,q''') \notin \mu} \bigwedge_{q'' \in Q} [(a, q', \alpha', q'', \notin) \vee (s, q'', -1, q''', \notin)]$$

In ϕ^3 the state q' appears in $target(\zeta_1)$ so it will have one rank only. For q'' we again take conjunction over all possible states. If we take a negative automaton $\overline{A}$ in state r, it either appears in ζ_2 with one rank only or does not appear at all. If the second is the case, we are done. If the first is the case then for this specific rank $(s_{(3,5)}, \overline{A}_r, -1, q''', \notin)$ will not hold. If it does not hold, $(a, q', \alpha', q''', \notin)$ has to hold. Since ζ_2 and μ_2 are consistent the only possible appearance of $\overline{A}_r$ in μ_2 is with the same specific rank i. So once again we do not have to mark the rank of q'', $(s_{(3,5)}, \overline{A}_r, -1, q''', \notin)$ is true if $\overline{A}_r$ does not appear at all in ζ_2 and $(a, q', \alpha', \overline{A}_r, \notin)$ is true if $\overline{A}_r$ does not appear at all in μ_2.

The state q''' is somewhat more complex. Suppose $q''' = \overline{A}_r$. Either $(q, \alpha, \overline{A}_r)$ does not belong to μ_1 for all ranks, or it does belong for exactly one rank. If the first is the case then we can define $(s_{(3,5)}, q'', -1, \overline{A}_r, \notin)$ as true if no rank of $\overline{A}_r$ appears in ζ_2. If some rank appears in ζ_2 then it must be the case that for the same q'', $(q', \alpha', q'') \notin \mu_2$. In the second case we record the rank of $\overline{A}_r$ and interpret $(s_{(3,5)}, q'', -1, \overline{A}_{(r,i)}, \notin)$ as true over ζ_2 if for other ranks $(q'', -1, \overline{A}_r)$ does not appear in ζ_2.

$$\phi^5(a, \zeta, \mu) = \bigwedge_{(q,1,q') \in \zeta} \bigwedge_{(q,\alpha,q'') \notin \mu} (s, q, -1, q'', \notin)$$

The analysis of ϕ^5 is similar. The state q appears in $state(\zeta)$ and the state q'' is similar to q''' in ϕ^3.

As the alphabet monitors the ranks of negative automata, in the book-keeping component we have to follow only states from $cl(g) \times \{\bot, \top\}$. It is enough to consider $\{-2, -1, 0, 1, 2\}^{cl(g)}$.

Concluding the last few paragraphs, given a formula g of length $|g| = n$, the nondeterministic Büchi automaton recognizing $L(g)$ has $2^{O(n^2 log(n))}$ states. Denote m as the maximal number of states of an automaton connective in the formula g. Let $K = \{-2m, ..., -0, 0, 1\} \cup \{nr\}$ with same semantics as before[2]. A state is a five tuple (q, C, S, A, P):

1. $q \in \{q_0, q_1\}$
2. $C \in K^{\{c\} \times rclg \times \{\in, \notin\}}$

[2] nr - A negative automaton with no rank, for conditions 3 and 5 of the annotation and for the consistency of the strategy

3. $S \in K^{\{s_{(3,5)}, s_{(4,6)}\} \times rcl(g) \times \{-1\} \times rcl(g) \times \{\notin\}}$

4. $A \in K^{\{a\} \times rcl(g) \times \{\bot, \top\} \times rcl(g) \times \{\in, \notin\}}$

5. $P \in \{-2, -1, 0, 1, 2\}^{rcl(g) \times \{\bot, \top\}}$

The initial states impose no conditions on the strategy and the annotation of the first letter (C, S, A consist of rank 0 only). The formula $(g, \bot)$ appears in P and no state in P appears with rank -2 or 2.

The acceptance set consists of all sets in which the book-keeping component in P is empty, i.e. no state in P is ranked -2 or 2.

The transition combines the transitions of B_1 and B_2 with the last results on the rank memorylessness.

Chapter 4

Conclusions

We have shown decision procedures for the logics ETL_f, ETL_l and ETL_r proposed by Vardi, Wolper and Sistla [WVS83]. Given an ETL_f (ETL_l) formula of length n we give a nondeterministic Büchi automaton with 3^n or 4^n states. The emptiness of this automaton can be checked in linear nondeterministic space. Given an ETL_r formula we build a nondeterministic Büchi automaton with $2^{O(n\ log(n))}$ states. The emptiness of this automaton can be checked in $O(n\ log(n))$ nondeterministic space.

We follow the suggestion of Wolper, Vardi and Sistla [WVS83] and augment temporal logic with alternating automata. Given an ATL formula of length n we build a nondeterministic automaton with $2^{O(n\ log(n))}$ states.

Our final move is to add 2-way alternating automata to the logic. Given a $2ATL$ formula of length n, its decision procedure is in $O(n^2 log(n))$ space. All three decision procedures are PSPACE-complete, the first (ETL) from [VW94] and the latter two subsume the first. A summary of these results can be found in Table 4.1.

Vardi [Var98] has shown how to convert 2-way alternating parity automata on trees to 1-way nondeterministic parity automata on trees. The method used in Section 3.6 can be used to handle 2-way parity automata on trees. Thus, given a 2-way alternating parity automaton we can give an equivalent 1-way alternating parity automaton (a projection of the language of the second is the language of the first).

Vardi's construction [Var98] includes strategy and annotation. The strategy is a way to satisfy the transition of the automaton and the annotation is a finite representation of 2-way run segments.

Formula of length n	2ABW	1ABW	NBW
ETL_f		$2n$ states	3^n / 4^n states
ETL_l		$2n$ states	3^n / 4^n states
ETL_r		$O(n^2)$ states	$2^{O(n\ log(n))}$ states
ETL_a		$O(n^2)$ states	$2^{O(n\ log(n))}$ states
ETL_{2a}	$O(n^2)$ states	$O(n^4)$ states	$2^{O(n^2 log(n))}$ states

Table 4.1: Summary of results

The annotation includes also information about future 2-way run segments. If we use alternating automata, the annotation can be restricted to data about the past alone (both on trees and on words). Removing this part of the annotation requires augmenting the construction with parts that check that such future run segments do exist. Such a part of the automaton will resemble the construction of the 1-way alternating automaton in Appendix A.

Bibliography

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

[Bir96] J.C. Birget. Two-way automata and length-preserving homomorphisms. *Mathematical Systems Theory*, 29(3):191–226, 1996.

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

[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.

[DFV99] N. Daniele, F.Guinchiglia, and M.Y. Vardi. Improved automata generation for linear temporal logic. In *Computer Aided Verification, Proc. 11th Int. Conference*, volume 1633 of *Lecture Notes in Computer Science*, pages 249–260. Springer-Verlag, 1999.

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

[EL87] E.A. Emerson and C.-L. Lei. Modalities for model checking: Branching time logic strikes back. *Science of Computer Programming*, 8:275–306, 1987.

[GPVW95] R. Gerth, D. Peled, M.Y. Vardi, and P. Wolper. Simple on-the-fly automatic verification of linear temporal logic. In P. Dembiski and M. Sredniawa, editors, *Protocol Specification, Testing, and Verification*, pages 3–18. Chapman & Hall, August 1995.

[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.

[HU87] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*, chapter 2, pages 13–45. Addison-Wesley Publishing Company, 1987.

[Isl96] A. Isli. Converting a Büchi alternating automaton to a usual nondeterministic one. *the Indian Journal SADHANA*, 21:213–228, 1996.

[Koz83] D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

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

[LPZ85] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In *Logics of Programs*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218, Brooklyn, June 1985. Springer-Verlag.

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

[Mic88] M. Michel. Complementation is more difficult with automata on infinite words. CNET, Paris, 1988.

[MP92] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, Berlin, January 1992.

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

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symposium on Foundation of Computer Science*, pages 46–57, 1977.

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

[Saf88] S. Safra. On the complexity of ω-automata. In *Proc. 29th IEEE Symposium on Foundations of Computer Science*, pages 319–327, White Plains, October 1988.

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

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

[SV89] S. Safra and M.Y. Vardi. On ω-automata and temporal logic. In *Proc. 21st ACM Symposium on Theory of Computing*, pages 127–137, Seattle, May 1989.

[SVW87] A.P. Sistla, M.Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *Theoretical Computer Science*, 49:217–237, 1987.

[Tho98] W. Thomas. Complementation of Büchi automata revisited. *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–122, 1998.

[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. Unified verification theory. In B. Banieqbal, H. Barringer, and A. Pnueli, editors, *Proc. Temporal Logic in Specification*, volume 398, pages 202–212. Lecture Notes in Computer Science, Springer-Verlag, 1989.

[Var96] M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In F. Moller and G. Birtwistle, editors, *Logics for Concurrency: Structure versus Automata*, volume 1043 of *Lecture Notes in Computer Science*, pages 238–266. Springer-Verlag, Berlin, 1996.

[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.

[VW94] M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, November 1994.

[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.

[Wol83] P. Wolper. Temporal logic can be more expressive. *Information and Control*, 56(1–2):72–99, 1983.

[WVS83] P. Wolper, M.Y. Vardi, and A.P. Sistla. Reasoning about infinite computation paths. In *Proc. 24th IEEE Symposium on Foundations of Computer Science*, pages 185–194, Tucson, 1983.

Appendix A

Converting 2-way nondeterministic automata to 1-way alternating automata

We give a construction for converting 2-way nondeterministic automata to 1-way alternating automata. We first give a construction for finite automata. We enhance our construction to apply to 2-way nondeterministic Büchi, parity and Rabin automata on infinite words.

We use the fact that the run of the nondeterministic automaton goes back and forth along the input word. We analyze the form such a run can take and recognize, using alternating automata, when such a run exists.

Given a 2-way nondeterministic automaton with n states we build an equivalent 1-way alternating automaton with $O(n^2)$ states. We also show how to build an alternating automaton that recognizes the complementary language with the same size. Vardi [Var88] converted a 2-way nondeterministic Büchi automaton directly into an exponential 1-way nondeterministic Büchi automaton. If we convert our alternating Büchi automata into a nondeterministic automata [MH84] we get automata of the same size as in [Var88].

A.1 Definitions

A *2-way nondeterministic automaton* is a five-tuple $A = \langle \Sigma, S, s_0, \delta, F \rangle$ where $\delta : S \times \Sigma \to 2^{S \times \{-1,0,1\}}$ is the transition function. We can run A either on finite words (*2-way nondeterministic finite automaton* or *2NFA* in short) or on infinite words (*2-way nondeterministic Büchi automaton* or *NBW* in 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, $q_0 = s_0$ 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 Büchi automata, a run is *accepting* if it visits $F \times \mathbb{N}$ infinitely often.

A *2-way nondeterministic parity (Rabin) automaton* is a five tuple $A = \langle \Sigma, S, s_0, \delta, \alpha \rangle$ where Σ, S, s_0 and δ are like before and $\alpha = \{F_0, ..., F_m\}$ is a subset of 2^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 is defined just like for a 2NBW. A run r of a parity automaton is accepting if there exists an even i, $0 \leq i \leq m$ such that r visits $F_i \times \mathbb{N}$ infinitely often and for all $i' \leq i$, we have that r visits $F_{i'} \times \mathbb{N}$ only finitely often. A run r of a Rabin automaton is accepting if there exists an i, $0 \leq i \leq m$ such that r visits $G_i \times \mathbb{N}$ infinitely often and $B_i \times \mathbb{N}$ only finitely often.

A *1-way alternating automaton* is a five tuple $B = \langle \Sigma, Q, s_0, \Delta, F \rangle$ where $\Delta : S \times \Sigma \to B^+(Q)$ is the transition function. Again we may run A on finite words (*1AFA*) or on infinite words (*1ABW*).

A *run* of B 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 B on an infinite word $w = w_0 w_1 ...$ is defined similarly as an infinite labeled tree. A run is *accepting* if all its infinite paths are labeled by F infinitely often.

A.2 Automata on Finite Words

We start by transforming automata that run on finite words. We then enhance our method to automata on infinite words.

Theorem A.2.1 *For every 2NFA $A = \langle \Sigma, S, s_0, \rho, F \rangle$ with n states, there exist 1AFAs B and B' with $O(n^2)$ states such that $L(B) = L(A)$ and $L(B') = \Sigma^* \setminus L(A)$.*

A.2.1 Removing 'zero' steps

A *0-step* 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 a 0-step if $i_j = i_{j-1}$.

Our first conversion is from $A = \langle \Sigma, S, s_0, \delta, F \rangle$ with $\delta : S \times \Sigma \to 2^{S \times \{-1,0,1\}}$ to an equivalent $A' = \langle \Sigma, S, s_0, \delta', F \rangle$ with $\delta' : S' \times \Sigma \to 2^{S \times \{-1,1\}}$. There are no 0-steps in the run of the second.

We start by defining for each state s and alphabet letter a, the set C_a^s of all states reachable from s with 0 steps using letter a. We call C_a^s the *0-closure* of s and a.

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

Define $\delta''(s, a) = \bigcup_{t \in C_a^s} \delta(t, a)$ 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 A.2.2 $L(A) = L(A')$

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.2.2 Two-way runs

From this point on we consider only 2NFAs with no 0-steps. We use $A = \langle \Sigma, S, s_0, \delta, F \rangle$ to denote the 2NFA and $B = \langle \Sigma, Q, s_0, \Delta, F \rangle$ to denote its equivalent 1AFA[1].

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.

We will be only interested in runs in which the same state in the same position do no repeat twice during the run.

Definition 7 Simple Run
A run $r = (s_0, 0), (s_1, i_1), (s_2, i_2), ..., (s_m, i_m)$ is simple if for all j and k such that $j < k$, either $s_j \neq s_k$ or $i_j \neq i_k$.

Claim A.2.1 *There exists an accepting run of A on w iff there exists a simple accepting run of A on w.*

Proof: A simple run is a run. Given an accepting run $r = (s_0, 0), (s_1, i_1), (s_2, i_2), ..., (s_m, i_m)$ 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 $(s_0, 0), ..., (s_j, i_j), (s_{k_1}, i_{k+1}), ..., (s_m, i_m)$. 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. □

Given the 2NFA A our goal is to construct the 1AFA B recognizing the same language. In Figure A.1a we see that a run of A takes the form of a 'zigzag'. Our one-way automaton will read words moving forward and accept if such a 'zigzag' run exists. In Figure A.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 will be $Q = S \cup (S \times S)$. A state $s \in Q$ represents a part of the run that is only looking forward (s_4 in Figure A.1a). A pair state $(s_1, s_3) \in Q$

[1]Note that B uses the acceptance set of A.

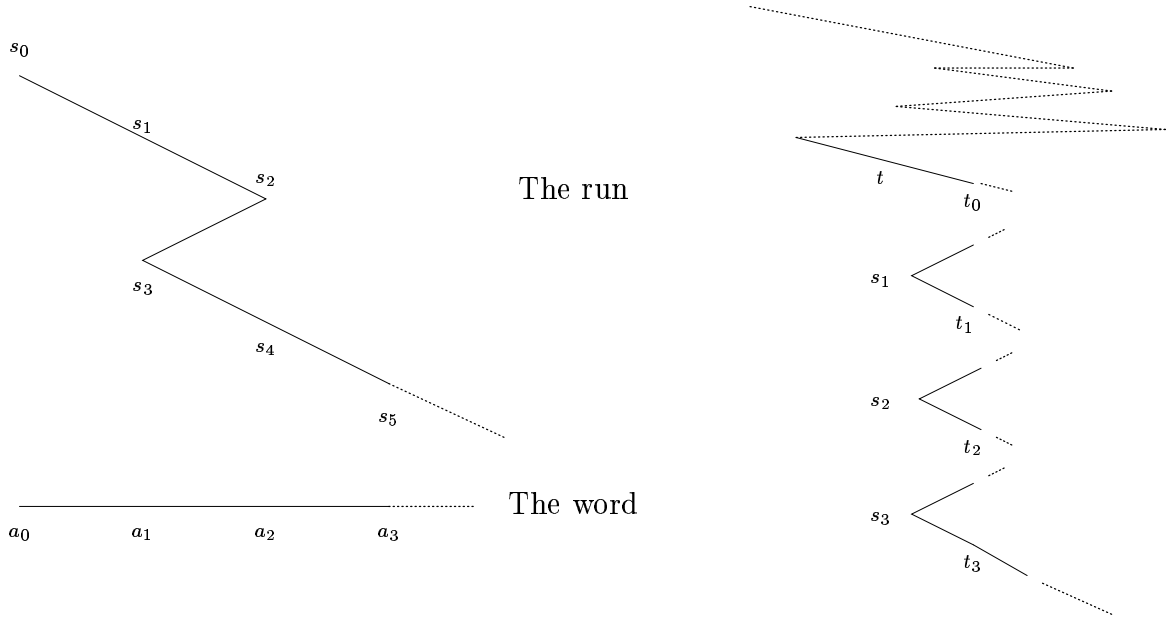

Figure A.1: (a) A zigzag run (b) The transition at the singleton state t

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 A.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 $(s_3, -1) \in \delta(s_2, a_2)$.

A.2.3 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 A.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 can not be larger than 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 $t_1, ..., t_k$. Hence the alternating automaton has to verify that there is a run segment connecting the successor of t (denoted t_0) to s_1 (we assume by induction that 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_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

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

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.

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

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 A.2a, the run segment connecting t to s might visit letter a_j but should not visit a_{j-1}.

Figure A.2b provides a more detailed example. The automaton in state (t,s) guesses that the run segment linking t to s visits a_2 twice. It guesses 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.

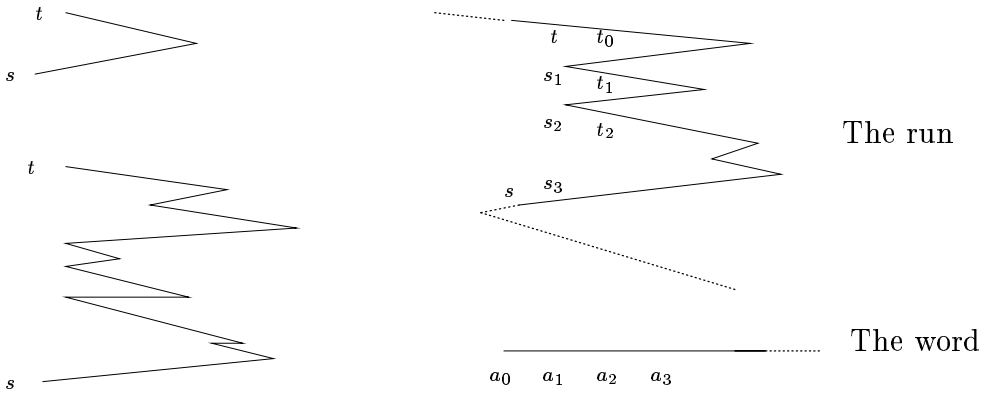

Figure A.2: (a) Different connecting segments (b) The transition at the pair state (t,s)

We now define the transition from 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 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 ($(t_i, 1) \in \delta(s_i, a)$).

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

The transition of B chooses one of these sequences and ensures that all pairs meet in due time:

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

A.2.4 Proof of correctness

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

Claim A.2.3 $L(A) = L(B)$

Proof: Given an accepting simple run of A on a word w of the form $(s_0, 0), (s_1, i_1), ..., (s_m, i_m)$ we annotate each pair by the place it took in the run of A. Thus the run takes the form $(s_0, 0, 0), (s_1, i_1, 1), ..., (s_m, i_m, m)$. We build a run tree of B by induction. In addition to labeling the nodes of the trees with states of B ($Q \cup Q \times Q$) we attach a single tag to a singleton state and a pair of tags to a pair state. The tag will be a triplet from the annotated run of A. For example the root of the run tree of B will be 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 A.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 $(t_k, i+1, j_k+1)$ will 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 there are no visits to letters before a_{i+1}.

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 A.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.

It is easy to see that all pair-labeled paths terminate with 'true' before reading the whole word w and the single path labeled by single 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. 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 $R_{a_0}^{s_0}$ ensures that $(p, 1) \in \delta(s, a_0)$. When we return to ϵ after scanning the whole tree the run of 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 already 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 $R_{a_{|x|}}^{(t,s)}$ 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 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. □

A.3 Automata on infinite words

In a first glance it seems that the exact same construction should work for Büchi automata. Eliminate the 0-steps and then just make sure that the single infinite path visits F infinitely often. This is not the case, there are two problems. For one the visits to F may be 'hidden'. For example consider the following run segment $..., (q,i), (q',i), (q'',i+1), ...$ In case q' is a member of F, we shrink the sequence to $..., (q,i), (q'',i+1), ...$ and reduce the number of visits to F by one. If we repeat this action infinitely many times we might turn an accepting run into a rejecting one. A similar problem occurs when checking that a 'zigzag run' exists, what if the visits to F are in the zigzags and not along the main path forward ? The second problem is loops that visit F. In the sequel we solve these problems.

Theorem A.3.1 *For every 2NBW $A = \langle \Sigma, S, s_0, \rho, F \rangle$ with n states, there exist 1ABWs B and B' with $O(n^2)$ states such that $L(B) = L(A)$ and $L(B') = \Sigma^\omega \setminus L(A)$.*

A.3.1 Zero steps

Given an automaton $A = \langle \Sigma, S, s_0, \delta, F \rangle$ where $\delta : S \times \Sigma \to 2^{S \times \{-1,0,1\}}$ we would like to remove all the 0-steps. There are two potential problems, visits to F in a 0-step and a loop of 0-steps that visits F. Hence we double the number of states and add an accepting sink state $A' = \langle \Sigma, (S \times \{\bot, \top\}) \cup \{Acc\}, (s_0, \bot), \delta', (S \times \{\top\}) \cup \{Acc\} \rangle$. A sequence like $..., ((s,\bot),i), ((s',\top),i+1), ...$ in the run means that in the run of A between the appearance of (s,i) and $(s',i+1)$ there was a 0-step that visited F. Similarly $\bot$ means that 0-steps (if occured) have not visited F (see also [Wil99, HK96] where 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 0-steps reading letter a and one last forward/backward step. All states avoid the acceptance set F.

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

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.

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

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 $1 \leq k$ 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 $0 \leq j \leq k$, 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.

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

Apparently, A' is 0-step free.

Claim A.3.2 *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 $\bot$ 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. □

A.3.2 Two-way runs

Once again we consider only 2NBWs with no 0-steps. As in the case of 0-steps there are two issues to be considered. Hidden visits to the accepting set F and loops.

If we take the alternating automaton we built in the finite case and simply run it on infinite words, we demand that the pair-labeled paths should be finite and that the infinite singleton-labeled path should 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 similar to the case of the loop in the 0-steps section. The automaton might be stuck forever in a finite prefix of the word w. We will show that in this case we can find an alternative accepting run of A in which the suffix of the run is of the form $(t_1, t_2, ..., t_m)^\omega$ where one of the states t_i is a member of F.

Once again we are interested in runs in which the same state in the same position do not repeat twice during the run. In an infinite run it might be impossible to avoid it completely, hence we try to minimize such events.

Definition 8 Simple Run
A run $r = (s_0, 0), (s_1, i_1), (s_2, i_2), ...$ *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* $i_j = i_{j+m}$.

Claim A.3.1 *There exists an accepting run of* A *on* w *iff there exists a simple accepting run of* A *on* w.

Proof: A simple run is a run. Given a run $r = (s_0, 0), (s_1, i_1), (s_2, i_2), ...$, 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 $(s_0, 0), ..., (s_{j-1}, i_{j-1}), ((s_j, i_j), ..., (s_{k-1}, i_{k-1}))^\omega$. 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. □

We use $A = \langle \Sigma, S, s_0, \delta, F \rangle$ to denote the 2NBW and $B = \langle \Sigma, Q, s'_0, \Delta, F' \rangle$ to denote the 1ABW. As mentioned, we have to record hidden visits to the set 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 $\bot$ and $\top$. Hence $Q = (S \cup S \times S) \times \{\bot, \top\}$. 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 singleton state $(s, \top)$ is displaying a visit to F in the zigzags connecting s to the previous singleton state.

The same notation enables us to solve the problem of loop. This is done by allowing a transition from a singleton state to a sequence of pair states and demanding that one of this pairs will promise a visit to F. Details follow.

Some of the unknowns in the definition of B are: $Q = (S \cup S \times S) \times \{\bot, \top\}$, $s'_0 = (s_0, \bot)$ and $F' = (S \times \{\top\})$. The transition function Δ is defined in the next section.

A.3.3 The Construction

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.

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

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

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

However this is not enough. We have to consider also the case of a loop. The automaton has to guess that the run will terminate 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 will be 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 $n+1^{\text{th}}$ 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 $(t_p, 1) \in \delta(s_{k+1}, a)$).

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

It is quite obvious that a visit to F has to occur within the loop. Hence given the sequence $< t_0, s_1, t_1, ..., s_k, t_k, s_{k+1} >$ 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 will visit 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 $p = 0$ also works in this case. Again one visit to F is enough hence all other pairs are annotated by $\bot$.

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

The transition of B has to choose one of the sequences in $R_a^t \cup L_a^t$. And then choose a sequence of $\bot$ and $\top$.

$$\Delta((t, \beta), a) = \bigvee \begin{array}{l} \bigvee_{R_a^t} \bigvee_{\alpha_k^R} (t_0, s_1, \alpha_0) \wedge (t_1, s_2, \alpha_1) \wedge ... \wedge (t_{k-1}, s_k, \alpha_{k-1}) \wedge (t_k, \alpha_k) \\ \\ \bigvee_{L_a^t} \bigvee_{\alpha_{k,p}^L} (t_0, s_1, \alpha_0) \wedge (t_1, s_2, \alpha_1) \wedge ... \wedge (t_k, s_{k+1}, \alpha_k) \end{array}$$

Where β is either $\bot$ or $\top$.

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

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

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 ($\bot$). Otherwise the promise must be passed to one of the successors.

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

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

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

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

A.3.4 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 A.3.3 *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 will work. 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 will be 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 $\top$ will appear 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 $(s_1, i, j_1), (s_2, i, j_2), ...,$ 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 $(t_l, i+1), ..., (s_{l+1}, i)$ will visit F. Annotate the pair (t_l, s_{l+1}) by $\top$ and all the others by $\bot$.

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 will end and we will be 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. □

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 0-steps. 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 $8n^2 + 4n$ states. We could actually unite the two stages of the construction into one stage. Such a construction will include the 0-steps 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 will result in an automaton with approximately $2n^2 + 2n$ states.

A.3.5 Complementing the alternating automaton

Complementing an ABW is not as easy as complementing an AFA. 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 A with n states, we constructed a 1ABW B with $O(n^2)$ states. If we implement the quadratic construction from [KV97] on B we get B', a 1ABW with $O(n^4)$ states accepting the complementary language of A. We show how to construct an 1ABW with $O(n^2)$ states whose language is the complement of A's language. We recall the proof in [KV97] and show how to avoid the quadratic price in our case. The following is taken from [KV97] with minor adjustments.

Definition 9 *[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 10 *[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 $E \subseteq \bigcup_{i=0}^{\infty}(Q \times \{i\}) \times (Q \times \{i+1\})$:
$V = \{(V(x), |x|) | x \in T\}$
$E = \{((V(x), |x|), (V(y), |y|)) | x, y \in T \text{ and } y \text{ successor of } x \text{ in } T\}$

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:

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

Definition 11 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 A.3.2 *[KV97] For every $i \geq 0$, either $w(G_{2i}) = 0$ or $w(G_{2i+2}) < w(G_{2i})$.*

In our case, we can partition the state set of B 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 indexes are in the range $[2n+2]$.

Denote our co-Büchi automaton by $B' = \langle \Sigma, Q, (s_0, \bot), \Delta', F \rangle$ where $Q = (S \cup S \times S) \times \{\bot, \top\}$. Kupferman and Vardi show that how to construct a weak alternating automaton with state set $Q \times \{0, ..., 2n+2\}$ that accepts the same language.

We can further reduce the number of states. We know that only pair-states are reachable from pair-states and that there is no pair-state 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 indexes 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 B' that is the complement of B. Given $B = \langle \Sigma, Q, (s_0, \bot), \Delta, F \rangle$ where $Q = (S \cup S \times S) \times \{\bot, \top\}$ we define $\overline{B} = \langle \Sigma, Q', s_0', \overline{\Delta}, F' \rangle$ 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 $\phi \in B^+(Q)$, and a rank $i \in [2n]$, the formula $release(\phi, i)$ is obtained from ϕ by replacing every element (s, α) from $S \times \{\bot, \top\}$ by $\bigvee_{l \leq i}(s, \alpha, l)$. Let Δ' be the dualization of Δ then:

$$\overline{\Delta}((s, \alpha, i), a) = \begin{cases} release(\Delta'((s, \alpha), a)) & \text{if } \alpha = \bot \text{ or } i \text{ is even} \\ false & \text{if } \alpha = \top \text{ and } i \text{ is odd} \end{cases}$$

$$\overline{\Delta}((s, t, \alpha), a) = \Delta'((s, t, \alpha), a)$$

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

A.3.6 Parity and Rabin acceptance conditions

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

Theorem A.3.4 *For every 2-way nondeterministic Rabin (parity) automaton $A = \langle \Sigma, S, s_0, \rho, \alpha \rangle$ with n states and index m, there exists a 1ABW B with $O(n^2 \cdot m)$ states such that $L(B) = L(A)$.*

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

This construction can be improved as following. Build a 1ABW B for A (without constructing A' first). Multiply the state set of B by the index (and one extra copy) $m+1$. The i^{th} copy of the automaton will avoid all the states in B_i. The alternating automaton starts running in copy 0. The transition at a singleton state in copy 0 will include also a guess whether to stay in copy 0 or guess that states from B_i will not be visited again during the run and then move to copy i. We should allow also moving into copy i in the middle of the transition into 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 i^{th} copy will include 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.

When given a Parity automaton one may convert it to a Rabin automaton and then apply the above modification. Taking care of Parity automata without reducing it to Rabin is also possible. The changes to the construction are very similar to the ones described above for Rabin automata.

A.4 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 new automaton is quadratic in the number of states of the 2-way automaton and the size of the transition is exponential in the size of the original transition. If we further convert our alternating automaton into a nondeterministic automaton we get an automaton with $2^{O(n^2)}$ states. Vardi [Var89] showed that given a 2-way nondeterministic automaton, it is possible to construct a 1-way nondeterministic automaton recognizing the complementary language with $2^{O(n)}$ states. Given a 2-way nondeterministic automaton and seeking an automaton that recognizes the complementary language one should obviously choose his construction.

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