Automata: from logics to algorithms*

Moshe Y. Vardi[1]
Thomas Wilke[2]

[1] Department of Computer Science
Rice University
6199 S. Main Street
Houston, TX 77005-1892, U.S.A.
vardi@cs.rice.edu

[2] Institut für Informatik
Christian-Albrechts-Universität zu Kiel
Christian-Albrechts-Platz 4
24118 Kiel, Germany
wilke@ti.informatik.uni-kiel.de

Abstract

We review, in a unified framework, translations from five different logics—monadic second-order logic of one and two successors (S1S and S2S), linear-time temporal logic (LTL), computation tree logic (CTL), and modal μ-calculus (MC)—into appropriate models of finite-state automata on infinite words or infinite trees. Together with emptiness-testing algorithms for these models of automata, this yields decision procedures for these logics. The translations are presented in a modular fashion and in a way such that optimal complexity bounds for satisfiability, conformance (model checking), and realizability are obtained for all logics.

1 Introduction

In his seminal 1962 paper [17], Büchi states: "Our results [...] may therefore be viewed as an application of the theory of finite automata to logic." He was referring to the fact that he had proved the decidability of the monadic-second order theory of the natural numbers with successor function by translating formulas into finite automata, following earlier work by himself [16], Elgot [35], and Trakhtenbrot [122]. Ever since, the approach these pioneers were following has been applied successfully in many different contexts and emerged as a major paradigm. It has not only brought about a number of decision procedures for mathematical theories, for instance, for the monadic second-order theory of the full binary tree [100],

* We are grateful to Detlef Kähler, Christof Löding, Oliver Matz, Damian Niwiński, and Igor Walukiewicz for comments on drafts of this paper.

Jörg Flum, Erich Grädel, Thomas Wilke (eds.). *Logic and Automata: History and Perspectives.* Texts in Logic and Games 2, Amsterdam University Press 2007, pp. 629–736.

but also efficient algorithms for problems in verification, such as a highly useful algorithm for LTL model checking [124].

The "automata-theoretic paradigm" has been extended and refined in various aspects over a period of more than 40 years. On the one hand, the paradigm has led to a wide spectrum of different models of automata, specifically tailored to match the distinctive features of the logics in question, on the other hand, it has become apparent that there are certain automata-theoretic constructions and notions, such as determinization of automata on infinite words [85], alternation [90], and games of infinite duration [12, 54], which form the core of the paradigm.

The automata-theoretic paradigm is a common thread that goes through many of Wolfgang Thomas's scientific works. In particular, he has written two influential survey papers on this topic [118, 120].

In this paper, we review translations from five fundamental logics, monadic second-order logic of one successor function (S1S), monadic second-order logic of two successor functions (S2S), linear-time temporal logic (LTL), computation tree logic (CTL), and the modal μ-calculus (MC) into appropriate models of automata. At the same time, we use these translations to present some of the core constructions and notions in a unified framework. While adhering, more or less, to the chronological order as far as the logics are concerned, we provide modern translations from the logics into appropriate automata. We attach importance to present the translations in a modular fashion, making the individual steps as simple as possible. We also show how the classical results on S1S and S2S can be used to derive first decidability results for the three other logics, LTL, CTL, and MC, but the focus is on how more refined techniques can be used to obtain good complexity results.

While this paper focuses on the translations from logics into automata, we refer the reader to the excellent surveys [118, 120] and the books [52, 96] for the larger picture of automata and logics on infinite objects and the connection with games of infinite duration.

Basic notation and terminology

Numbers. In this paper, the set of natural numbers is denoted ω, and each natural number stands for the set of its predecessors, that is, $n = \{0, \ldots, n-1\}$.

Words. An alphabet is a nonempty finite set, a word over an alphabet A is a function $n \to A$ where $n \in \omega$ for a finite word and $n = \omega$ for an infinite word. When $u\colon n \to A$ is a word, then n is called its length and denoted $|u|$, and, for every $i < n$, the value $u(i)$ is the letter of u in position i. The set of all finite words over a given alphabet A is denoted A^*, the set of all infinite words over A is denoted A^ω, the empty word is denoted ε, and A^+

stands for $A^* \setminus \{\varepsilon\}$.

When u is a word of length n and $i, j \in \omega$ are such that $0 \leq i, j < n$, then $u[i,j] = u(i) \dots u(j)$, more precisely, $u[i,j]$ is the word u' of length $\max\{j - i + 1, 0\}$ defined by $u'(k) = u(i+k)$ for all $k < |u'|$. In the same fashion, we use the notation $u[i,j)$. When u denotes a finite, nonempty word, then we write $u(*)$ for the last letter of u, that is, when $|u| = n$, then $u(*) = u(n-1)$. Similarly, when u is finite or infinite and $i < |u|$, then $u[i,*)$ denotes the suffix of u starting at position i.

Trees. In this paper, we deal with trees in various contexts, and depending on these contexts we use different types of trees and model them in one way or another. All trees we use are directed trees, but we distinguish between trees with unordered successors and n-ary trees with named successors.

A tree with unordered siblings is, as usual, a tuple $\mathscr{T} = (V, E)$ where V is a nonempty set of vertices and $E \subseteq V \times V$ is the set of edges satisfying the usual properties. The root is denoted $\mathrm{root}(\mathscr{T})$, the set of successors of a vertex v is denoted $\mathrm{sucs}^{\mathscr{T}}(v)$, and the set of leaves is denoted $\mathrm{lvs}(\mathscr{T})$.

Let n be a positive natural number. An n-ary tree is a tuple $\mathscr{T} = (V, \mathrm{suc}_0, \dots, \mathrm{suc}_{n-1})$ where V is the set of vertices and, for every $i < n$, suc_i is the ith successor relation satisfying the condition that for every vertex there is at most one ith successor (and the other obvious conditions). Every n-ary tree is isomorphic to a tree where V is a prefix-closed nonempty subset of n^* and $\mathrm{suc}_i(v, v')$ holds for $v, v' \in V$ iff $v' = vi$. When a tree is given in this way, simply by its set of vertices, we say that the tree is given in implicit form. The full binary tree, denoted $\mathscr{T}_{\mathrm{bin}}$, is 2^* and the full ω-tree is ω^*. In some cases, we replace n in the above by an arbitrary set and speak of D-branching trees. Again, D-branching trees can be in implicit form, which means they are simply a prefix-closed subset of D^*.

A branch of a tree is a maximal path, that is, a path which starts at the root and ends in a leaf or is infinite. If an n-ary tree is given in implicit form, a branch is often denoted by its last vertex if it is finite or by the corresponding infinite word over n if it is infinite.

Given a tree $\mathscr{T}$ and a vertex v of it, the subtree rooted at v is denoted $\mathscr{T}{\downarrow}v$.

In our context, trees often have vertex labels and in some rare cases edge labels too. When L is a set of labels, then an L-labeled tree is a tree with a function l added which assigns to each vertex its label. More precisely, for trees with unordered successors, an L-labeled tree is of the form (V, E, l) where $l\colon V \to E$; an L-labeled n-ary tree is a tuple $(V, \mathrm{suc}_0, \dots, \mathrm{suc}_{n-1}, l)$ where $l\colon V \to E$; an L-labeled n-ary tree in implicit form is a function $t\colon V \to L$ where $V \subseteq n^*$ is the set of vertices of the tree; an L-labeled D-branching tree in implicit form is a function $t\colon V \to L$ where $V \subseteq D^*$ is the set of vertices of the tree. Occasionally, we also have more than one

vertex labeling or edge labelings, which are added as other components to the tuple.

When $\mathscr{T}$ is an L-labeled tree and u is a path or branch of $\mathscr{T}$, then the labeling of u in $\mathscr{T}$, denoted $l^{\mathscr{T}}(u)$, is the word w over L of the length of u and defined by $w(i) = l(u(i))$ for all $i < |u|$.

Tuple Notation. Trees, graphs, automata and the like are typically described as tuples and denoted by calligraphic letters such as $\mathscr{T}$, $\mathscr{G}$, and so on, possibly furnished with indices or primed. The individual components are referred to by $V^{\mathscr{T}}$, $E^{\mathscr{G}}$, $E^{\mathscr{T}'}$, ... The ith component of a tuple $t = (c_0, \ldots, c_{r-1})$ is denoted $\mathrm{pr}_i(t)$.

2 Monadic second-order logic of one successor

Early results on the close connection between logic and automata, such as the Büchi–Elgot–Trakhtenbrot Theorem [16, 35, 122] and Büchi's Theorem [17], center around monadic second-order logic with one successor relation (S1S) and its weak variant (WS1S). The formulas of these logics are built from atomic formulas of the form $\mathrm{suc}(x, y)$ for first-order variables x and y and $x \in X$ for a first-order variable x and a set variable (monadic second-order variable) X using boolean connectives, first-order quantification ($\exists x$), and second-order quantification for sets ($\exists X$). The two logics differ in the semantics of the set quantifiers: In WS1S quantifiers only range over finite sets rather than arbitrary sets.

S1S and WS1S can be used in different ways. First, one can think of them as logics to specify properties of the natural numbers. The formulas are interpreted in the structure with the natural numbers as universe and where suc is interpreted as the natural successor relation. The most important question raised in this context is:

Validity. Is the (weak) monadic second-order theory of the natural numbers with successor relation decidable? (Is a given sentence valid in the natural numbers with successor relation?)

A slightly more general question is:

Satisfiability. Is it decidable whether a given (W)S1S formula is satisfiable in the natural numbers?

This is more general in the sense that a positive answer for closed formulas only already implies a positive answer to the first question. Therefore, we only consider satisfiability in the following.

Second, one can think of S1S and WS1S as logics to specify the behavior of devices which get, at any moment in time, a fixed number of bits as input and produce a fixed number of bits as output (such as sequential circuits), see Figure 1. Then the formulas are interpreted in the same structure as above, but for every input bit and for every output bit there will be exactly

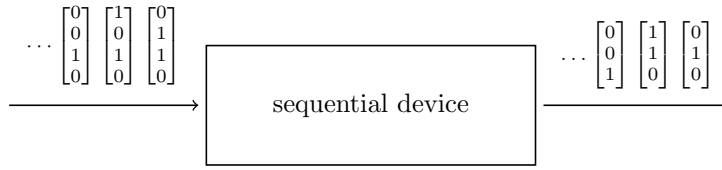


FIGURE 1. Sequential device

one free set variable representing the moments in time where the respective bit is true. (The domain of time is assumed discrete; it is identified with the natural numbers.) A formula will then be true for certain input-output pairs—coded as variable assignments—and false for the others.

For instance, when we want to specify that for a given device with one input bit, represented by the set variable X, and one output bit, represented by Y, it is the case that for every other moment in time where the input bit is true the output bit is true in the subsequent moment in time, we can use the following formula:

$$\exists Z(\text{“}Z \text{ contains every other position where } X \text{ is true”} \wedge \forall x(x \in Z \rightarrow \text{“the successor of } x \text{ belongs to } Y\text{”})).$$

That the successor of x belongs to Y is expressed by $\forall y(\mathrm{suc}(x, y) \rightarrow y \in Y)$. That Z contains every other position where X is true is expressed by the conjunction of the three following conditions, where we assume, for the moment, that the “less than” relation on the natural numbers is available:

- Z is a subset of X, which can be stated as $\forall x(x \in Z \rightarrow x \in X)$,
- If X is nonempty, then the smallest element of X does not belong to Z, which can be stated as $\forall x(x \in X \wedge \forall y(y < x \rightarrow \neg y \in X) \rightarrow \neg x \in Z)$.
- For all $x, y \in X$ such that $x < y$ and such that there is no element of X in between, either x or y belongs to Z, which can be stated as

$$\forall x \forall y(x \in X \wedge y \in X \wedge x < y \wedge \forall z(x < z \wedge z < y \rightarrow \neg z \in X) \rightarrow (x \in Z \leftrightarrow \neg y \in Z)).$$

To conclude the example, we need a formula that specifies that x is less than y. To this end, we express that y belongs to a set which does not contain x but with each element its successor:

$$\exists X(\neg x \in X \wedge \forall z \forall z'(z \in X \wedge \mathrm{suc}(z, z') \rightarrow z' \in X) \wedge y \in X).$$

The most important questions that are raised with regard to this usage of (W)S1S are:

Conformance. Is it decidable whether the input-output relation of a given device satisfies a given formula?

Realizability. Is it decidable whether for a given formula there exists a device with an input-output relation satisfying the formula (and if so, can a description of such a device be produced effectively)?

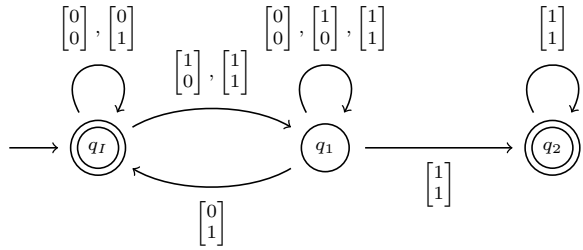
Obviously, it is important what is understood by "device". For instance, Church, when he defined realizability in 1957 [23], was interested in boolean circuits. We interpret device as "finite-state device", which, on a certain level of abstraction, is the same as a boolean circuit.

In this section, we first describe Büchi's Theorem (Section 2.1), from which we can conclude that the first two questions, satisfiability and conformance, have a positive answer. The proof of Büchi's Theorem is not very difficult except for a result about complementing a certain type of automaton model for infinite words, which we then establish (Section 2.2). After that we prove a result about determinization of the same type of automaton model (Section 2.3), which serves as the basis for showing that realizability is decidable, too. The other ingredient of this proof, certain games of infinite duration, are then presented, and finally the proof itself is given (Section 2.4).

2.1 Büchi's Theorem

The connection of S1S and WS1S to automata theory, more precisely, to the theory of formal languages, is established via a simple observation. Assume that φ is a formula such that all free variables are set variables among $V_0, \ldots, V_{m-1}$, which we henceforth denote by $\varphi = \varphi(V_0, \ldots, V_{m-1})$. Then the infinite words over $[2]_m$, the set of all column vectors of height m with entries from $\{0, 1\}$, correspond in a one-to-one fashion to the variable assignments $\alpha\colon \{V_0, \ldots, V_{m-1}\} \to 2^\omega$, where 2^M stands for the power set of any set M. More precisely, for every infinite word $u \in [2]_m^\omega$ let α_u be the variable assignment defined by $\alpha_u(V_j) = \{i < \omega\colon u(i)_{[j]} = 1\}$, where, for every $a \in [2]_m$, the expression $a_{[j]}$ denotes entry j of a. Then α_u ranges over all variable assignments as u ranges over all words in $[2]_m^\omega$. As a consequence, we use $u \models \varphi$, or, when "weak quantification" (only finite sets are considered) is used, $u \models^{\mathrm{w}} \varphi$ rather than traditional notation such as $\mathfrak{N}, \alpha \models \varphi$ (where $\mathfrak{N}$ stands for the structure of the natural numbers). Further, when φ is a formula as above, we define two formal languages of infinite words depending on the type of quantification used:

$$\mathscr{L}(\varphi) = \{u \in [2]_m^\omega\colon u \models \varphi\}, \qquad \mathscr{L}^{\mathrm{w}}(\varphi) = \{u \in [2]_m^\omega\colon u \models^{\mathrm{w}} \varphi\}.$$



The initial state has an incoming edge without origin; final states are shown as double circles.

FIGURE 2. Example for a Büchi automaton

We say that φ defines the language $\mathscr{L}(\varphi)$ and weakly defines the language $\mathscr{L}^{\mathrm{w}}(\varphi)$. Note that, for simplicity, the parameter m is not referred to in our notation.

Büchi's Theorem states that the above languages can be recognized by an appropriate generalization of finite-state automata to infinite words, which we introduce next. A Büchi automaton is a tuple

$$\mathscr{A} = (A, Q, Q_I, \Delta, F)$$

where A is an alphabet, Q is a finite set of states, $Q_I \subseteq Q$ is a set of initial states, $\Delta \subseteq Q \times A \times Q$ is a set of transitions of $\mathscr{A}$, also called its transition relation, and $F \subseteq Q$ is a set of final states of $\mathscr{A}$. An infinite word $u \in A^\omega$ is accepted by $\mathscr{A}$ if there exists an infinite word $r \in Q^\omega$ such that $r(0) \in Q_I$, $(r(i), u(i), r(i+1)) \in \Delta$ for every i, and $r(i) \in F$ for infinitely many i. Such a word r is called an accepting run of $\mathscr{A}$ on u. The language recognized by $\mathscr{A}$, denoted $\mathscr{L}(\mathscr{A})$, is the set of all words accepted by $\mathscr{A}$.

For instance, the automaton in Figure 2 recognizes the language corresponding to the formula

$$\forall x(x \in V_0 \to \exists y(x < y \wedge y \in V_1)),$$

which says that every element from V_0 is eventually followed by an element from V_1. In Figure 2, q_I is the state where the automaton is not waiting for anything; q_1 is the state where the automaton is waiting for an element from V_1 to show up; q_2 is used when from some point onwards all positions belong to V_0 and V_1. Nondeterminism is used to guess that this is the case.

Büchi's Theorem can formally be stated as follows.

Theorem 2.1 (Büchi, [17])**.**

1. There exists an effective procedure that given a formula $\varphi = \varphi(V_0, \ldots, V_{m-1})$ outputs a Büchi automaton $\mathscr{A}$ such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\varphi)$
2. There exists an effective procedure that given a Büchi automaton $\mathscr{A}$ over an alphabet $[2]_m$ outputs a formula $\varphi = \varphi(V_0, \ldots, V_{m-1})$ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathscr{A})$.

The proof of part 2 is straightforward. The formula which needs to be constructed simply states that there exists an accepting run of $\mathscr{A}$ on the word determined by the assignment to the variables V_i. One way to construct φ is to write it as $\exists X_0 \ldots \exists X_{n-1} \psi$ where each set variable X_i corresponds exactly to one state of $\mathscr{A}$ and where ψ is a first-order formula (using $<$ in addition to suc) which states that the X_i's encode an accepting run of the automaton (the X_i's must form a partition of ω and the above requirements for an accepting run must be satisfied): 0 must belong to one of the sets X_i representing the initial states; there must be infinitely many positions belonging to sets representing final states; the states assumed at adjacent positions must be consistent with the transition relation.

The proof of part 1 is more involved, although the proof strategy is simple. The desired automaton $\mathscr{A}$ is constructed inductively, following the structure of the given formula. First-order variables, which need to be dealt with in between, are viewed as singletons. The induction base is straightforward and two of the three cases to distinguish in the inductive step are so, too: disjunction on the formula side corresponds to union on the automaton side and existential quantification corresponds to projection. For negation, however, one needs to show that the class of languages recognized by Büchi automata is closed under complementation. This is not as simple as with finite state automata, especially since deterministic Büchi automata are strictly weaker than nondeterministic ones, which means complementation cannot be done along the lines known from finite words.

In the next subsection, we describe a concrete complementation construction.

Büchi's Theorem has several implications, which all draw on the following almost obvious fact. Emptiness for Büchi automata is decidable. This is easy to see because a Büchi automaton accepts a word if and only if in its transition graph there is a path from an initial state to a strongly connected component which contains a final state. (This shows that emptiness can even be checked in linear time and in nondeterministic logarithmic space.)

Given that emptiness is decidable for Büchi automata, we can state that the first question has a positive answer:

Corollary 2.2 (Büchi, [17])**.** Satisfiability is decidable for S1S.

Proof. To check whether a given S1S formula $\varphi = \varphi(V_0, \ldots, V_{m-1})$ is satisfiable one simply constructs the Büchi automaton which is guaranteed to exist by Büchi's Theorem and checks this automaton for non-emptiness. Q.E.D.

Observe that in the above corollary we use the term "satisfiability" to denote the decision problem (Given a formula, is it satisfiable?) rather than the question from the beginning of this section (Is it decidable whether ...). For convenience, we do so in the future too: When we use one of the terms satisfiability, conformance, or realizability, we refer to the corresponding decision problem.

For conformance, we first need to specify formally what is meant by a finite-state device, or, how we want to specify the input-output relation of a finite-state device. Remember that we think of a device as getting inputs from $[2]_m$ and producing outputs from $[2]_n$ for given natural numbers m and n. So it is possible to view an input-output relation as a set of infinite words over $[2]_{m+n}$. To describe an entire input-output relation of a finite-state device we simply use a nondeterministic finite-state automaton. Such an automaton is a tuple

$$\mathscr{D} = (A, S, S_I, \Delta)$$

where A is an alphabet, S is a finite set of states, $S_I \subseteq S$ is a set of initial states, and $\Delta \subseteq S \times A \times S$ is a transition relation, just as with Büchi automata. A word $u \in A^\omega$ is accepted by $\mathscr{D}$ if there exists $r \in S^\omega$ with $r(0) \in S_I$ and $(r(i), u(i), r(i+1)) \in \Delta$ for every $i < \omega$. The set of words accepted by $\mathscr{D}$, denoted $\mathscr{L}(\mathscr{D})$, is the language recognized by $\mathscr{D}$. Observe that $\mathscr{L}(\mathscr{D})$ is exactly the same as the language recognized by the Büchi automaton which is obtained from $\mathscr{D}$ by adding the set S as the set of final states.

Conformance can now be defined as follows: Given an S1S formula $\varphi = \varphi(X_0, \ldots, X_{m-1}, Y_0, \ldots, Y_{n-1})$ and a finite-state automaton $\mathscr{D}$ with alphabet $[2]_{m+n}$, determine whether $u \models \varphi$ for all $u \in \mathscr{L}(\mathscr{D})$.

There is a simple approach to decide conformance. We construct a Büchi automaton that accepts all words $u \in \mathscr{L}(\mathscr{D})$ which do not satisfy the given specification φ, which means we construct a Büchi automaton which recognizes $\mathscr{L}(\mathscr{D}) \cap \mathscr{L}(\neg\varphi)$, and check this automaton for emptiness. Since Büchi's Theorem tells us how to construct an automaton $\mathscr{A}$ that recognizes $\mathscr{L}(\neg\varphi)$, we only need a construction which, given a finite-state automaton $\mathscr{D}$ and a Büchi automaton $\mathscr{A}$, recognizes $\mathscr{L}(\mathscr{A}) \cap \mathscr{L}(\mathscr{D})$. The construction

The product of a Büchi automaton $\mathscr{A}$ and a finite-state automaton $\mathscr{D}$, both over the same alphabet A, is the Büchi automaton denoted $\mathscr{A} \times \mathscr{D}$ and defined by

$$\mathscr{A} \times \mathscr{D} = (A, Q \times S, Q_I \times S_I, \Delta, F \times S)$$

where

$$\Delta = \{((q,s),a,(q',s')) \colon (q,a,q') \in \Delta^{\mathscr{A}} \text{ and } (s,a,s') \in \Delta^{\mathscr{D}}\}.$$

FIGURE 3. Product of a Büchi automaton with a finite-state automaton

depicted in Figure 3, which achieves this, is a simple automata-theoretic product. Its correctness can be seen easily.

Since we already know that emptiness is decidable for Büchi automata, we obtain:

Corollary 2.3 (Büchi, [17])**.** Conformance is decidable for S1S.

From results by Stockmeyer and Meyer [112, 111], it follows that the complexity of the two problems from Corollaries 2.2 and 2.3 is nonelementary, see also [102].

Another immediate consequence of Büchi's Theorem and the proof of part 2 as sketched above is a normal form theorem for S1S formulas. Given an arbitrary S1S formula, one uses part 1 of Büchi's Theorem to turn it into an equivalent Büchi automaton and then part 2 to reconvert it to a formula. The proof of part 2 of Büchi's Theorem is designed in such a way that a formula will emerge which is of the form $\exists V_0 \ldots \exists V_{n-1} \psi$ where ψ is without second-order quantification but uses $<$. Such formulas are called existential S1S formulas.

Corollary 2.4 (Büchi-Thomas, [17, 117])**.** Every S1S formula is equivalent to an existential S1S formula, moreover, one existential set quantifier is sufficient.

To conclude this subsection we note that using the theory of finite automata on finite words only, one can prove a result weaker than Büchi's Theorem. In the statement of this theorem, automata on finite words are used instead of Büchi automata and the weak logic is used instead of the full logic. Moreover, one considers only variable assignments for the free set variables that assign finite sets only. The latter is necessary to be able to

describe satisfying assignments by finite words. Such a result was obtained independently by Büchi [16], Elgot [35], and Trakhtenbrot [122], preceding Büchi's work on S1S.

2.2 Complementation of Büchi automata

Büchi's original complementation construction, more precisely, his proof of the fact that the complement of a language recognized by a Büchi automaton can also be recognized by a Büchi automaton, as given in [17], follows an algebraic approach. Given a Büchi automaton $\mathscr{A}$, he defines an equivalence relation on finite words which has

1. only a finite number of equivalence classes and
2. the crucial property that $UV^\omega \subseteq \mathscr{L}(\mathscr{A})$ or $UV^\omega \cap \mathscr{L}(\mathscr{A}) = \varnothing$ for all its equivalence classes U and V.

Here, UV^ω stands for the set of all infinite words which can be written as $uv_0v_1v_2\ldots$ where $u \in U$ and $v_i \in V$ for every $i < \omega$. To complete his proof Büchi only needs to show that

(a) each set UV^ω is recognized by a Büchi automaton,

(b) every infinite word over the given alphabet belongs to such a set, and

(c) the class of languages recognized by Büchi automata is closed under union.

To prove (b), Büchi uses a weak variant of Ramsey's Theorem; (a) and (c) are easy to see. The equivalence relation Büchi defines is similar to Nerode's congruence relation. For a given word u, he considers

(i) all pairs (q, q') of states for which there exists a path from q to q' labeled u and

(ii) all pairs (q, q') where, in addition, such a path visits a final state,

and he defines two nonempty finite words to be equivalent if they agree on these pairs. If one turns Büchi's "complementation lemma" into an actual complementation construction, one arrives at a Büchi automaton of size $2^{\theta(n^2)}$ where n denotes the number of states of the given Büchi automaton.

Klarlund [65] and Kupferman and Vardi [74] describe complementation constructions along the following lines. Given a Büchi automaton $\mathscr{A}$ and a word u over the same alphabet, they consider the run DAG of $\mathscr{A}$ on u, which is a narrow DAG which contains exactly the runs of $\mathscr{A}$ on u. Vertices in this run DAG are of the form (q, i) with $q \in Q$ and $i \in \omega$ and all runs where the ith state is q visit this vertex. They show that u is not accepted

by $\mathscr{A}$ if and only if the run DAG can be split into at most $2n$ alternating layers of two types where within the layers of the first type every vertex has proper descendants which are labeled with nonfinal states and where within the layers of the second type every vertex has only a finite number of descendants (which may be final or nonfinal). This can easily be used to construct a Büchi automaton for the complement: It produces the run DAG step by step, guesses for each vertex to which layer it belongs, and checks that its guesses are correct. To check the requirement for the layers of the second type, it uses the Büchi acceptance condition. The size of the resulting automaton is $2^{\theta(n \log n)}$. Optimizations lead to a construction with $(0.97n)^n$ states [46], while the best known lower bound is $(0.76n)^n$, established by Yan [131]. For practical implementations of the construction by Kupferman and Vardi, see [55].

In Section 2.2.2, we describe a complementation construction which is a byproduct of the determinization construction we explain in Section 2.3. Both constructions are based on the notion of reduced acceptance tree, introduced by Muller and Schupp [91] and described in what follows.

2.2.1 Reduced acceptance trees

Recall the notation and terminology with regard to binary trees introduced in Section 1.

Let $\mathscr{A}$ be a Büchi automaton as above, u an infinite word over the alphabet A. We consider a binary tree, denoted $\mathscr{T}_u$, which arranges all runs of $\mathscr{A}$ on u in a clever fashion, essentially carrying out a subset construction that distinguishes between final and nonfinal states, see Figure 4 for a graphical illustration.

The tree $\mathscr{T}_u$ is given as $l_u \colon V_u \to 2^Q$ in implicit form and defined inductively as follows.

(i) $\varepsilon \in V_u$ and $l_u(\varepsilon) = Q_I$.

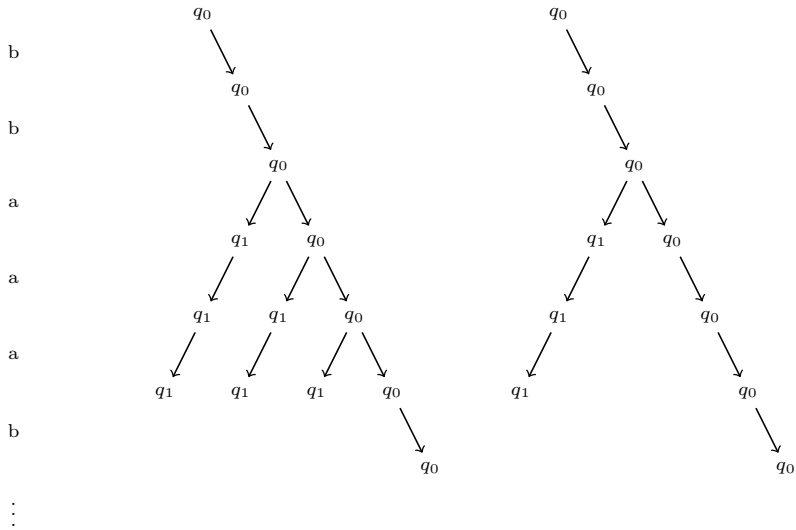
(ii) Let $v \in V_u$, $Q' = l_u(v)$, $a = u(|v|)$, and $Q'' = \bigcup\{\Delta(q,a) \colon q \in Q'\}$. Here and later, we use $\Delta(q,a)$ to denote $\{q' \in Q \colon (q,a,q') \in \Delta\}$.

- If $Q'' \cap F \neq \varnothing$, then $v0 \in V_u$ and $l_u(v0) = Q'' \cap F$.
- If $Q'' \setminus F \neq \varnothing$, then $v1 \in V_u$ and $l_u(v1) = Q'' \setminus F$.

The resulting tree is called the run tree of u with respect to $\mathscr{A}$.

A partial run of $\mathscr{A}$ on u is a word $r \in Q^+ \cup Q^\omega$ satisfying $r(0) \in Q_I$ and $(r(i), u(i), r(i+1)) \in \Delta$ for all i such that $i+1 < |r|$. A run is an infinite partial run.

Every partial run r of $\mathscr{A}$ on u determines a path b in the run tree: The length of b is $|r| - 1$ and $b(i) = 0$ if $r(i+1) \in F$ and $b(i) = 1$ otherwise, for $i < |r| - 1$. We write $r{\downarrow\downarrow}$ for this path and call it the 2-projection of r.



Depicted are the run tree and the reduced run tree of the automaton to the right for the given word. Note that in the trees the labels of the vertices should be sets of states, but for notational convenience we only list their elements.

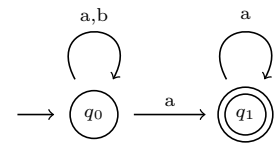


FIGURE 4. Run tree and reduced run tree

Clearly, if r is an accepting run of $\mathscr{A}$ on u, then $r{\downarrow\downarrow}$ has infinitely many left turns, where a left turn is a vertex which is a left successor. Conversely, if b is an infinite branch of $\mathscr{T}_u$, then there exists a run r of $\mathscr{A}$ on u such that $r{\downarrow\downarrow} = b$, and if b has infinitely many left turns, then r is accepting. This follows from Kőnig's lemma.

From this, we can conclude:

Remark 2.5. An infinite word u is accepted by a Büchi automaton $\mathscr{A}$ if and only if its run tree has a branch with an infinite number of left turns. We call such a branch an acceptance witness.

The tree $\mathscr{T}_u$ has two other interesting properties, which we discuss next. The first one is that $\mathscr{T}_u$ has a "left-most" acceptance witness, provided there is one at all. This acceptance witness, denoted b_u, can be constructed as follows. Inductively, assume $b_u(i)$ has already been defined for all $i < n$ in a way such that there is an acceptance witness with prefix $b' = b_u(0) \dots b_u(n-$

1). If there is an acceptance witness with prefix $b'0$, we set $b_u(n) = 0$. Otherwise, there must be an acceptance witness with prefix $b'1$, and we set $b_u(n) = 1$. Clearly, this construction results in an acceptance witness. One can easily prove that b_u is the left-most acceptance witness in the sense that it is minimal among all acceptance witnesses with respect to the lexicographical ordering (but we do not need this here).

The second interesting property says something about the states occurring to the left of b_u. We say a state q is persistent in a vertex v of a branch b of $\mathscr{T}_u$ if there is a run r of $\mathscr{A}$ on u such that $r{\downarrow\downarrow} = b$ and $q \in r(|v|)$, in other words, q is part of a run whose 2-projection contains v. A word $v \in \{0,1\}^*$ is said to be left of a word $w \in \{0,1\}^*$, denoted $v <_{\text{lft}} w$, if $|v| = |w|$ and $v <_{\text{lex}} w$, where $<_{\text{lex}}$ denotes the lexicographical ordering. The crucial property of b_u is:

Lemma 2.6. Let u be an infinite word accepted by a Büchi automaton $\mathscr{A}$, w a vertex on the left-most acceptance witness b_u, and q a state which is persistent in w on b_u. Then $q \notin l_u(v)$ for every $v \in V_u$ such that $v <_{\text{lft}} w$.

Proof for reduced split tree that we can keep only the leftmost occurrence of a state

Proof. Assume that w is a vertex on b_u and that $v \in V_u$ is left of w, let $n = |v|$ $(= |w|)$. For contradiction, assume q is persistent in w on b_u and $q \in l_u(v) \cap l_u(w)$. Since $q \in l_u(v)$, we know there is a partial run r of $\mathscr{A}$ on u with $r{\downarrow\downarrow} = v$ and $r(n) = q$.

Since q is persistent in w on b_u there exists a run r' of $\mathscr{A}$ on u such that $r'{\downarrow\downarrow} = b_u$ and $r'(n) = q$. Then $r'[n,\infty)$ is an uninitialized run of $\mathscr{A}$ on $u[n,\infty)$ starting with q, where an uninitialized run is one where it is not required that the first state is the initial state. This implies that $r'' = rr'(n,\infty)$ is a run of $\mathscr{A}$ on u. Moreover, $r(i) = r''(i)$ for all $i \geq n$, which implies $r''{\downarrow\downarrow}$ is an acceptance witness, too. Let c be the longest common prefix of $r''{\downarrow\downarrow}$ and b_u. We know that $c0 \leq_{\text{prf}} r''{\downarrow\downarrow}$ and $c1 \leq_{\text{prf}} b_u$, which is a contradiction to the definition of b_u—recall that $r''{\downarrow\downarrow}$ is an acceptance witness. Q.E.D.

The above fact can be used to prune $\mathscr{T}_u$ in such a way that it has finite width, but still contains an acceptance witness if and only if u is accepted by $\mathscr{A}$. We denote the pruned tree by $\mathscr{T}'_u$, write it as $l'_u\colon V'_u \to 2^Q$), and call it the reduced acceptance tree. Informally, $\mathscr{T}'_u$ is obtained from $\mathscr{T}_u$ by keeping on each level only the first occurrence of a state, reading the level from left to right, see Figure 4. Formally, the reduced acceptance tree is inductively defined as follows.

(i) $\varepsilon \in V'_u$ and $l'_u(\varepsilon) = Q_I$.

(ii) Let $v \in V'_u$, $Q' = l'_u(v)$, $a = u(|v|)$, and $Q'' = \bigcup\{\Delta(q,a)\colon q \in Q'\}$, just as above. Assume $l'_u(w)$ has already been defined for $w <_{\text{lft}} v0$ and let $\bar{Q} = \bigcup\{l'_u(w)\colon w \in V'_u \text{ and } w <_{\text{lft}} v0\}$.

- If $Q'' \cap F \setminus \bar{Q} \neq \varnothing$, then $v0 \in V'_u$ and $l'_u(v0) = Q'' \cap F \setminus \bar{Q}$.
- If $Q'' \setminus (F \cup \bar{Q}) \neq \varnothing$, then $v1 \in V'_u$ and $l'_u(v1) = Q'' \setminus (F \cup \bar{Q})$.

As a consequence of Lemma 2.6, we have:

Corollary 2.7. Let $\mathscr{A}$ be a Büchi automaton and u an infinite word over the same alphabet. Then $u \in \mathscr{L}(\mathscr{A})$ iff $\mathscr{T}'_u$ contains an acceptance witness.

Since $\mathscr{T}'_u$ is a tree of width at most $|Q|$, it has at most $|Q|$ infinite branches. So u is not accepted by $\mathscr{A}$ if and only if there is some number n such that $b(i)$ is not a left turn for all infinite branches b of $\mathscr{T}'_u$. This fact can be used to construct a Büchi automaton for the complement language, as is shown in what follows.

... all infinite branches contain only a finite number of left turns

...after some number n there are only right turns

2.2.2 The complementation construction

Implemented in GOAL as slice -p

Let n be an arbitrary natural number and $v_0 <_{\text{lft}} v_1 <_{\text{lft}} \cdots <_{\text{lft}} v_{r-1}$ be such that $\{v_0, \ldots, v_{r-1}\} = \{v \in V'_u \colon |v| = n\}$, that is, $v_0, \ldots, v_{r-1}$ is the sequence of all vertices on level n of $\mathscr{T}'_u$, from left to right. We say that $l'_u(v_0) \ldots l'_u(v_{r-1})$, which is a word over the alphabet 2^Q, is slice n of $\mathscr{T}'_u$.

successor function

Slice n consists of two levels of the reduced split tree: level n and the edges to the vertices of level n+1

It is straightforward to construct slice $n+1$ from slice n, simply by applying the transition relation to each element of slice n and removing multiple occurrences of states just as with the construction of $\mathscr{T}'_u$. Suppose $Q_0 \ldots Q_{r-1}$ is slice n and $a = u(n)$. Let $Q'_0, \ldots, Q'_{2r-1}$ be defined by

$$Q'_{2i} = \Delta(Q_i, a) \cap F \setminus \bar{Q}_i, \qquad Q'_{2i+1} = \Delta(Q_i, a) \setminus (F \cup \bar{Q}_i),$$

where $\bar{Q}_i = \bigcup_{j<2i} Q'_j$. Further, let $j_0 < j_1 < \cdots < j_{s-1}$ be such that $\{j_0, \ldots, j_{s-1}\} = \{j < 2r \colon Q'_j \neq \varnothing\}$. Then $Q'_{j_0} \ldots Q'_{j_{s-1}}$ is slice $n+1$ of $\mathscr{T}'_u$. This is easily seen from the definition of the reduced run tree.

We say that a tuple $U = Q_0 \ldots Q_{r-1}$ is a slice over Q if $\varnothing \neq Q_i \subseteq Q$ holds for $i < r$ and if $Q_i \cap Q_j = \varnothing$ for all $i, j < r$ with $i \neq j$. The sequence $Q'_{j_0} \ldots Q'_{j_{s-1}}$ from above is said to be the successor slice for U and a and is denoted by $\delta_{\text{slc}}(Q_0 \ldots Q_{r-1}, a)$.

The automaton for the complement of $\mathscr{L}(\mathscr{A})$, denoted $\mathscr{A}^{\mathsf{C}}$, works as follows. First, it constructs slice after slice as it reads the given input word. We call this the initial phase. At some point, it guesses

(i) that it has reached slice n or some later slice, with n as described right after Corollary 2.7, and

(ii) which components of the slice belong to infinite branches.

The rest of its computation is called the repetition phase. During this phase it carries out the following process, called verification process, over and over again. It continues to construct slice after slice, checking that (i) the

components corresponding to vertices on infinite branches all continue to the right (no left turn anymore) and (ii) the components corresponding to the other branches die out (do not continue forever). The newly emerging components corresponding to branches which branch off to the left from the vertices on the infinite branches are marked. As soon as all branches supposed to die out have died out, the process starts all over again, now with the marked components as the ones that are supposed to die out.

To be able to distinguish between components corresponding to infinite branches, branches that are supposed to die out, and newly emerging branches, the components of the slice tuples are decorated by *inf*, *die*, or *new*. Formally, a decorated slice is of the form $(Q_0 \dots Q_{r-1}, f_0 \dots f_{r-1})$ where $Q_0 \dots Q_{r-1}$ is a slice and $f_i \in \{inf, die, new\}$ for $i < r$. A decorated slice where $f_i \neq die$ for all $i < r$ is called final.

The definition of the successor of a decorated slice is slightly more involved than for ordinary slices, and such a successor may not even exist. Assume a decorated slice as above is given, let V stand for the entire slice and U for its first component (which is an ordinary slice). Let the Q'_j's and j_i's be defined as above. The successor slice of V with respect to a, denoted $\delta_{\mathrm{d}}(V, a)$, does not exist if there is some $i < r$ such that $Q'_{2i+1} = \varnothing$ and $f_i = inf$, because this means that a branch guessed to be infinite and without left turn dies out. In all other cases, $\delta_{\mathrm{d}}(V, a) = (\delta_{\mathrm{slc}}(U, a), f'_{j_0} \dots f'_{j_{s-1}})$ where the f'_j's are defined as follows, depending on whether the automaton is within the verification process (V is not final) or at its end (V is final):

Slice V is not final. Then $f'_{2i} = f'_{2i+1} = f_i$ for every $i < r$, except when $f_i = inf$. In this case, $f'_{2i} = new$ and $f'_{2i+1} = f_i$.

Slice V is final. Then $f'_{2i} = f'_{2i+1} = die$ for every $i < r$, except when $f_i = inf$. In this case, $f'_{2i+1} = inf$ and $f'_{2i} = die$.

These choices reflect the behavior of the automaton as described above.

To describe the transition from the first to the second phase formally, assume U is a slice and $a \in A$. Let $\Delta_s(U, a)$ contain all decorated slices $(\delta_{\mathrm{slc}}(U, a), f_0 \dots f_{s-1})$ where $f_i \in \{inf, die\}$ for $i < s$. This reflects that the automaton guesses that certain branches are infinite and that the others are supposed to die out. The full construction of $\mathscr{A}^{\mathsf{C}}$ as outlined in this section is described in Figure 5. A simple upper bound on its number of states is $(3n)^n$.

Using L^{C} to denote the complement of a language, we can finally state:

Theorem 2.8. Let $\mathscr{A}$ be a Büchi automaton with n states. Then $\mathscr{A}^{\mathsf{C}}$ is a Büchi automaton with $(3n)^n$ states such that $\mathscr{L}(\mathscr{A}^{\mathsf{C}}) = \mathscr{L}(\mathscr{A})^{\mathsf{C}}$.

Let $\mathscr{A}$ be a Büchi automaton. The Büchi automaton $\mathscr{A}^{\mathsf{C}}$ is defined by

$$\mathscr{A}^{\mathsf{C}} = (A, Q^{\mathrm{s}} \cup Q^{\mathrm{d}}, Q_I, \Delta', F')$$

where the individual components are defined as follows:

$$\begin{aligned} Q^{\mathrm{s}} &= \text{set of slices over } Q, \\ Q^{\mathrm{d}} &= \text{set of decorated slices over } Q, \\ F' &= \text{set of final decorated slices over } Q, \end{aligned}$$

and where for a given $a \in A$ the following transitions belong to Δ':

- $(U, a, \delta_{\mathrm{slc}}(U, a))$ for every $U \in Q^{\mathrm{s}}$,
- (U, a, V) for every $U \in Q^{\mathrm{s}}$ and $V \in \Delta_{\mathrm{s}}(U, a)$,
- $(V, a, \delta_{\mathrm{d}}(V, a))$ for every $V \in Q^{\mathrm{d}}$, provided $\delta_{\mathrm{d}}(V, a)$ is defined.

FIGURE 5. Complementing a Büchi automaton

2.3 Determinization of Büchi automata

As noted above, determinstic Büchi automata are strictly weaker than nondeterministic ones in the sense that there are ω-languages that can be recognized by a nondeterministic Büchi automaton but by no deterministic Büchi automaton. (Following classical terminology, a Büchi automaton is called deterministic if $|Q_I| = 1$ and there is a function $\delta\colon Q \times A \to Q$ such that $\Delta = \{(q, a, \delta(q, a))\colon a \in A \wedge q \in Q\}$.) It turns out that this is due to the weakness of the Büchi acceptance condition. When a stronger acceptance condition—such as the parity condition—is used, every nondeterministic automaton can be converted into an equivalent deterministic automaton.

The determinization of Büchi automata has a long history. After a flawed construction had been published in 1963 [89], McNaughton, in 1966 [85], was the first to prove that every Büchi automaton is equivalent to a deterministic Muller automaton, a model of automata on infinite words with an acceptance condition introduced in Muller's work. In [43, 42], Emerson and Sistla described a determinization construction that worked only

for a subclass of all Büchi automata. Safra [105] was the first to describe a construction which turns nondeterministic Büchi automata into equivalent deterministic Rabin automata—a model of automata on infinite words with yet another acceptance condition—which has optimal complexity in the sense that the size of the resulting automaton is $2^{\theta(n \log n)}$ and one can prove that this is also a lower bound [86]. In 1995, Muller and Schupp [91] presented a proof of Rabin's Theorem via an automata-theoretic construction which has an alternative determinization construction with a similar complexity built-in; Kähler [76] was the first to isolate this construction, see also [1]. Kähler [76] also showed that based on Emerson and Sistla's construction one can design another determinization construction for all Büchi automata which yields automata of size $2^{\theta(n \log n)}$, too. In 2006, Piterman [97] showed how Safra's construction can be adapted so as to produce a parity automaton of the same complexity.

The determinization construction described below is obtained by applying Piterman's improvement of Safra's construction to Muller and Schupp's determinization construction. We first introduce parity automata, then continue our study of the reduced acceptance tree, and finally describe the determinization construction.

2.3.1 Parity automata

A parity automaton is very similar to a Büchi automaton. The only difference is that a parity automaton has a more complex acceptance condition, where every state is assigned a natural number, called priority, and a run is accepting if the minimum priority occurring infinitely often (the limes inferior) is even. States are not just accepting or rejecting; there is a whole spectrum. For instance, when the smallest priority is even, then all states with this priority are very similar to accepting states in Büchi automata: If a run goes through these states infinitely often, then it is accepting. When, on the other hand, the smallest priority is odd, then states with this priority should be viewed as being the opposite of an accepting state in a Büchi automaton: If a run goes through these states infinitely often, the run is not accepting. So parity automata allow for a finer classification of runs with regard to acceptance and rejection.

Formally, a parity automaton is a tuple

$$\mathscr{A} = (A, Q, Q_I, \Delta, \pi)$$

where A, Q, Q_I, and Δ are as with Büchi automata, but π is a function $Q \to \omega$, which assigns to each state its priority. Given an infinite sequence r of states of this automaton, we write $\mathrm{val}_\pi(r)$ for the limes inferior of the sequence $\pi(r(0)), \pi(r(1)), \ldots$ and call it the value of the run with respect to π. Since Q is finite, the value of each run is a natural number. A run

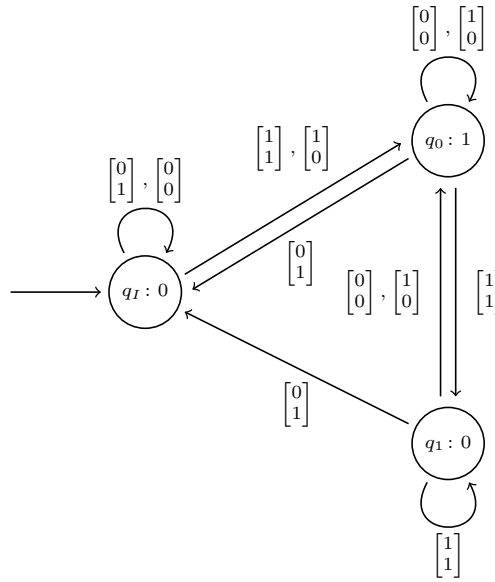


FIGURE 6. Deterministic parity automaton. The values in the circles next to the names of the states are the priorities.

r of $\mathscr{A}$ is accepting if its value is even. In other words, a run r of $\mathscr{A}$ is accepting if there exists an even number v and a number k such that

(i) $\pi(r(j)) \geq v$ for all $j \geq k$ and

(ii) $\pi(r(j)) = v$ for infinitely many $j \geq k$.

Consider, for example, the parity automaton depicted in Figure 6. It recognizes the same language as the Büchi automaton in Figure 2.

As far as nondeterministic automata are concerned, Büchi automata and parity automata recognize the same languages. On the one hand, every Büchi automaton can be viewed as a parity automaton where priority 1 is assigned to every non-final state and priority 0 is assigned to every final state. (That is, the parity automaton in Figure 6 can be regarded as a deterministic Büchi automaton.) On the other hand, it is also easy to see that every language recognized by a parity automaton is recognized by some Büchi automaton: The Büchi automaton guesses a run of the parity automaton and an even value for this run and checks that it is indeed the value of the run. To this end, the Büchi automaton runs in two phases. In the first phase, it simply simulates the parity automaton. At some point, it

Let $\mathscr{A}$ be a parity automaton. The Büchi automaton A^{par} is defined by

$$A^{\mathrm{par}} = (A, Q \cup Q \times E, Q_I, \Delta \cup \Delta', \{(q,k)\colon \pi(q) = k\})$$

where $E = \{\pi(q)\colon q \in Q \wedge \pi(q) \bmod 2 = 0\}$ and Δ' contains

- $(q, a, (q', k))$ for every $(q, a, q') \in \Delta$, provided $k \in E$, and
- $((q,k), a, (q', k))$ for every $(q, a, q') \in \Delta$, provided $\pi(q') \geq k$ and $k \in E$.

FIGURE 7. From parity to Büchi automata

concludes the first phase, guesses an even value, and enters the second phase during which it continues to simulate the parity automaton but also verifies (i) and (ii) from above. To check (i), the transition relation is restricted appropriately. To check (ii), the Büchi acceptance condition is used. This leads to the construction displayed in Figure 7. The state space has two different types of states: the states from the given Büchi automaton for the first phase and states of the form (q,k) where $q \in Q$ and k is a priority for the second phase. The priority in the second component never changes; it is the even value that the automaton guesses.

Remark 2.9. Let $\mathscr{A}$ be a parity automaton with n states and k different even priorities. Then the automaton A^{par} is an equivalent Büchi automaton with $(k+1)n$ states.

2.3.2 Approximating reduced run trees

Let $\mathscr{A}$ be a Büchi automaton as above and $u \in A^\omega$ an infinite word. The main idea of Muller and Schupp's determinization construction is that the reduced acceptance tree, $\mathscr{T}'_u$, introduced in Section 2.2.1, can be approximated by a sequence of trees which can be computed by a deterministic finite-state automaton. When these approximations are adorned with additional information, then from the sequence of the adorned approximations one can read off whether there is an acceptance witness in the reduced acceptance tree, which, by Remark 2.5, is enough to decide whether u is accepted.

For a given number n, the nth approximation of $\mathscr{T}'_u$, denoted $\mathscr{T}^n_u$, is the subgraph of $\mathscr{T}'_u$ which consists of all vertices of distance at most n from the

root and which are on a branch of length at least n. Only these vertices can potentially be on an infinite branch of $\mathscr{T}'_u$. Formally, $\mathscr{T}^n_u$ is the subtree of $\mathscr{T}'_u$ consisting of all vertices $v \in V'_u$ such that there exists $w \in V'_u$ satisfying $v \leq_{\mathrm{prf}} w$ and $|w| = n$, where $\leq_{\mathrm{prf}}$ denotes the prefix order on words.

Note that from Lemma 2.6 we can conclude:

Remark 2.10. When u is accepted by $\mathscr{A}$, then for every n the prefix of length n of b_u is a branch of $\mathscr{T}^n_u$.

The deterministic automaton to be constructed will observe how approximations evolve over time. There is, however, the problem that, in general, approximations grow as n grows. But since every approximation has at most $|Q|$ leaves, it has at most $|Q| - 1$ internal vertices with two successors—all other internal vertices have a single successor. This means that their structure can be described by small trees of bounded size, and only their structure is important, except for some additional information of bounded size. This motivates the following definitions.

A segment of a finite tree is a maximal path where every vertex except for the last one has exactly one successor, that is, it is a sequence $v_0 \dots v_r$ such that

(i) the predecessor of v_0 has two successors or v_0 is the root,

(ii) v_i has exactly one successor for $i < r$, and

(iii) v_r has exactly two successors or is a leaf.

Then every vertex of a given finite tree belongs to exactly one segment.

A contraction of a tree is obtained by merging all vertices of a segment into one vertex. Formally, a contraction of a finite tree $\mathscr{T}$ in implicit form is a tree $\mathscr{C}$ together with a function $c\colon V^{\mathscr{T}} \to V^{\mathscr{C}}$, the contraction map, such that the following two conditions are satisfied:

(i) For all $v, w \in V^{\mathscr{T}}$, $c(v) = c(w)$ iff v and w belong to the same segment.

When p is a segment of $\mathscr{T}$ and v one of its vertices, we write $c(p)$ for $c(v)$ and we say that $c(v)$ represents p.

(ii) For all $v \in V^{\mathscr{T}}$ and $i < 2$, if $vi \in V^{\mathscr{T}}$ and $c(v) \neq c(vi)$, then $\mathrm{suc}^{\mathscr{C}}_1(c(v), c(vi))$.

Note that this definition can easily be adapted to the case where the given tree is not in implicit form.

We want to study how approximations evolve over time. Clearly, from the nth to the $(n+1)$st approximation of $\mathscr{T}'_u$ segments can disappear, several segments can be merged into one, new segments of length one can

emerge, and segments can be extended by one vertex. We reflect this in the corresponding contractions by imposing requirements on the domains of consecutive contractions.

A sequence $\mathscr{C}_0, \mathscr{C}_1, \ldots$ of contractions with contraction maps $c_0, c_1, \ldots$ is a contraction sequence for u if the following holds for every n:

(i) $\mathscr{C}_n$ is a contraction of the nth approximation of $\mathscr{T}'_u$.

(ii) Let p and p' be segments of $\mathscr{T}_u^n$ and $\mathscr{T}_u^{n+1}$, respectively. If p is a prefix of p' (including $p = p'$), then $c_{n+1}(p') = c_n(p)$ and p' is called an extension of p in $n+1$.

(iii) If p' is a segment of $\mathscr{T}_u^{n+1}$ which consists of vertices not belonging to $\mathscr{T}$, then $c_{n+1}(p') \notin V^{\mathscr{C}_n}$, where $V^{\mathscr{C}_n}$ denotes the set of vertices of $\mathscr{C}_n$.

Since we are interested in left turns, we introduce one further notion. Assume that p and p' are segments of $\mathscr{T}_u^n$ and $\mathscr{T}_u^{n+1}$, respectively, and p is a prefix of p', just as in (ii) above. Let p'' be such that $p' = pp''$. We say that $c_{n+1}(p')$ (which is equal to $c_n(p)$) is left extending in $n+1$ if there is a left turn in p''.

For a graphical illustration, see Figure 8.

We can now give a characterization of acceptance in terms of contraction sequences.

Lemma 2.11. Let $\mathscr{C}_0, \mathscr{C}_1, \ldots$ be a contraction sequence for an infinite word u with respect to a Büchi automaton $\mathscr{A}$. Then the following are equivalent:

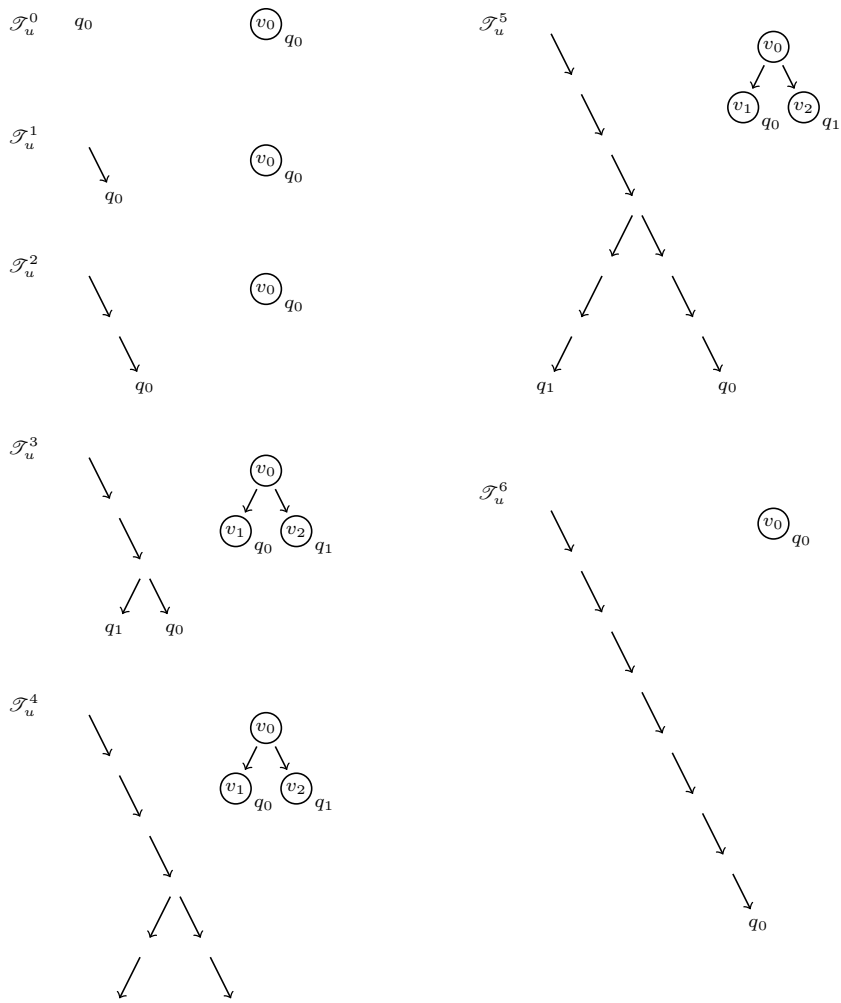
(A) $\mathscr{A}$ accepts u.

(B) There is a vertex v such that

  (a) $v \in V^{\mathscr{C}_n}$ for almost all n and

  (b) v is left extending in infinitely many n.

Proof. For the implication from (A) to (B), we start with a definition. We say that a segment p of the nth approximation is part of b_u, the left-most acceptance witness, if there are paths p_0 and p_1 such that $b_u = p_0 p p_1$. We say a vertex v represents a part of b_u if there exists i such that for all $j \geq i$ the vertex v belongs to $V^{\mathscr{C}_j}$ and the segment represented by v is part of b_u. Observe that from Remark 2.10 we can conclude that the root of $\mathscr{C}_0$ is such a vertex (where we can choose $i = 0$). Let V be the set of all vertices that represent a part of b_u and assume i is chosen such that $v \in V^{\mathscr{C}_j}$ for all $j \geq i$ and all $v \in V$. Then all elements from V form the same path in every $\mathscr{C}_j$ for $j \geq i$, say $v_0 \ldots v_r$ is this path.



Depicted is the beginning of the contraction sequence for $u = bbaaab\ldots$ with respect to the automaton from Figure 4. Note that, just as in Figure 4, we simply write q_i for $\{q_i\}$.

FIGURE 8. Contraction sequence

If the segment representing v_r is infinitely often extended, it will also be extended by a left turn infinitely often (because b_u is an acceptance witness), so v_r will be left extending in infinitely many i.

So assume that v_r is not extended infinitely often and let $i' \geq i$ be such that the segment represented by v_r is not extended any more for $j \geq i'$. Consider $\mathscr{C}_{i'+1}$. Let v' be the successor of v_r such that the segment represented by v' is part of b_u, which must exist because of Remark 2.10. Clearly, for the same reason, v' will be part of $V^{\mathscr{C}_j}$ for $j \geq i'+1$, hence $v' \in V$—a contradiction.

For the implication from (B) to (A), let v be a vertex as described in (B), in particular, let i be such that $v \in V^{\mathscr{C}_j}$ for all $j \geq i$. For every $j \geq i$, let p^j be the segment represented by v in $\mathscr{C}_j$. Since $p^i \leq_{\text{prf}} p^{i+1} \leq_{\text{prf}} p^{i+2} \leq_{\text{prf}} \dots$ we know there is a vertex w such that every p^j, for $j \geq i$, starts with w. Since the number of left turns on the p^j's is growing we know there is an infinite path d starting with w such that $p^j \leq_{\text{prf}} d$ for every $j \geq i$ and such that d is a path in $\mathscr{T}'_u$ with infinitely many left turns. The desired acceptance witness is then given by the concatenation of the path from the root to w, the vertex w itself excluded, and d. Q.E.D.

2.3.3 Muller–Schupp trees

The only thing which is left to do is to show that a deterministic finite-state automaton can construct a contraction sequence for a given word u and that a parity condition is strong enough to express (2.11) from Lemma 2.11. It turns out that when contractions are augmented with additional information, they can actually be used as the states of such a deterministic automaton. This leads us to the definition of Muller–Schupp trees.

Before we get to the definition of these trees, we observe that every contraction has at most $|Q|$ leaves, which means it has at most $2|Q|-1$ vertices. From one contraction to the next in a sequence of contractions, at most $|Q|$ new leaves—and thus at most $|Q|$ new vertices—can be introduced. In other words:

Remark 2.12. For every infinite word u, there is a contraction sequence $\mathscr{C}_0, \mathscr{C}_1, \dots$ such that $V^{\mathscr{C}_i} \subseteq V$ for every i for the same set V with $3|Q|$ vertices, in particular, $V = \{0, \dots, 3|Q|-1\}$ works.

A Muller-Schupp tree for $\mathscr{A}$ is a tuple

$$\mathscr{M} = (\mathscr{C}, l_q, l_l, R, h)$$

where

- $\mathscr{C}$ is a contraction with $V^{\mathscr{C}} \subseteq \{0, \dots, 3\,|Q|-1\}$,
- $l_q \colon \text{lvs}(\mathscr{C}) \to 2^Q$ is a leaf labeling,

- $l_l \colon V^{\mathscr{C}} \to \{0,1,2\}$ is a left labeling,
- $R \in \{0,\dots,3|Q|-1\}^*$ is a latest appearance record, a word without multiple occurrences of letters, and
- $h \leq |R|$ is the hit number.

To understand the individual components, assume $\mathscr{C}_0, \mathscr{C}_1, \dots$ is a contraction sequence for u with $V^{\mathscr{C}_n} \subseteq \{0,\dots,3|Q|-1\}$ for every n. (Recall that Remark 2.12 guarantees that such a sequence exists.) The run of the deterministic automaton on u to be constructed will be a sequence $\mathscr{M}_0, \mathscr{M}_1, \dots$ of Muller-Schupp trees $\mathscr{M}_n = (\mathscr{C}_n, l^n_q, l^n_l, R^n, h^n)$, such that the following conditions are satisfied, where c_n denotes the contraction map for $\mathscr{C}_n$:

Leaf labeling. For every n and every leaf $v \in \mathrm{lvs}(\mathscr{T}^n_u)$, the labeling of v will be the same as the labeling of the vertex of the segment representing the segment of this leaf, that is, $l^n_q(c_n(v)) = l'_u(v)$.

Left labeling. For every n and every $v \in V^{\mathscr{C}_n}$:

(i) if v represents a segment without left turn, then $l_n(v) = 0$,

(ii) if v is left extending in n, then $l^n_l(v) = 2$, and

(iii) $l^n_l(v) = 1$ otherwise.

Clearly, this will help us to verify (b) from Lemma 2.11(2.11).

Latest appearance record. The latest appearance record R^n gives us the order in which the vertices of $\mathscr{C}_n$ have been introduced. To make this more precise, for every n and $v \in V^{\mathscr{C}_n}$, let

$$d_n(v) = \min\{i \colon v \in V^{\mathscr{C}_j} \text{ for all } j \text{ such that } i \leq j \leq n\}$$

be the date of introduction of v. Then R^n is the unique word $v_0 \dots v_{r-1}$ over $V^{\mathscr{C}_n}$ without multiple occurrences such that

- $\{v_0,\dots,v_{r-1}\} = V^{\mathscr{C}_n}$,
- either $d_n(v_j) = d_n(v_k)$ and $v_j < v_k$ or $d_n(v_j) < d_n(v_k)$, for all j and k such that $j < k < r$.

We say that $v \in V^{\mathscr{C}_n}$ has index j if $R^n(j) = v$.

Hit number. The hit number h^n gives us the number of vertices whose index has not changed. Let $R^n = v_0 \dots v_{r-1}$ as above. The value h^n is the maximum number $\leq r$ such that $d_n(v_j) < n$ for $j < h$. In other words, the hit number gives us the length of the longest prefix of R^n which is a prefix of R^{n-1}.

We need one more definition before we can state the crucial property of Muller–Schupp trees. Let $\mathscr{M}$ be any Muller–Schupp tree as above and m the minimum index of a vertex with left labeling 2 (it is left extending). If such a vertex does not exist, then, by convention, $m = n$. We define $\pi(\mathscr{M})$, the priority of $\mathscr{M}$, as follows. If $m < h$, then $\pi(\mathscr{M}) = 2m$, and else $\pi(\mathscr{M}) = 2h + 1$.

Lemma 2.13. Let $\mathscr{A}$ be a Büchi automaton, u a word over the same alphabet, and $\mathscr{M}_0, \mathscr{M}_1, \ldots$ a sequence of Muller–Schupp trees satisfying the above requirements (leaf labeling, left labeling, latest appearance record, hit number). Let $p^\infty = \mathrm{val}_\pi(\mathscr{M}_0\mathscr{M}_1\ldots)$, that is, the smallest value occurring infinitely often in $\pi(\mathscr{M}_0)\pi(\mathscr{M}_1)\ldots$. Then the following are equivalent:

(A) $\mathscr{A}$ accepts u.

(B) p^∞ is even.

Proof. For the implication from (A) to (B), let v be a vertex as guaranteed by (B) in Lemma 2.11. There must be some n and some number i such that $v = R^n(i) = R^{n+1}(i) = \ldots$ and $R^n[0,i] = R^{n+1}[0,i] = \ldots$. This implies $h^j > i$ for all $j \geq n$, which means that if p^j is odd for some $j \geq n$, then $p^j > 2i$. In addition, since v is left extending for infinitely many j, we have $p^j \leq 2i$ and even for infinitely many j. Thus, p^∞ is an even value (less than or equal to $2i$).

For the implication from (B) to (A), assume that p^∞ is even and n is such that $p^j \geq p^\infty$ for all $j \geq n$. Let $n' \geq n$ be such that $p^{n'} = p^\infty$ and let v be the vertex of $\mathscr{C}_{n'}$ which gives rise to $p^{n'}$ (left extending with minimum index). Then $v \in V^{\mathscr{C}_j}$ for all $j \geq n'$ and v has the same index in all these $\mathscr{C}_j$. That is, whenever $p^j = p^\infty$ for $j \geq n'$, then v is left extending. So (B) from Lemma 2.11 is satisfied and we can conclude that u is accepted by $\mathscr{A}$. Q.E.D.

2.3.4 The determinization construction

Text

In order to arrive at a parity automaton, we only need to convince ourselves that a deterministic automaton can produce a sequence $\mathscr{M}_0, \mathscr{M}_1, \ldots$ as above. We simply describe an appropriate transition function, that is, we assume a Muller–Schupp tree $\mathscr{M}$ and a letter a are given, and we describe how $\mathscr{M}'$ is obtained from $\mathscr{M}$ such that if $\mathscr{M} = \mathscr{M}_n$ and $a = u(n)$, then $\mathscr{M}' = \mathscr{M}_{n+1}$. This is, in principle, straightforward, but it is somewhat technical. One of the issues is that during the construction of $\mathscr{M}'$ we have trees with more than $3\,|Q|$ vertices. This is why we assume that we are also given a set W of $2\,|Q|$ vertices disjoint from $\{0, \ldots, 3\,|Q| - 1\}$.

A Muller-Schupp tree $\mathscr{M}'$ is called an a-successor of $\mathscr{M}$ if it is obtained from $\mathscr{M}$ by applying the following procedure.

(i) Let $V_{\text{new}} = \{0, \ldots, 3\,|Q| - 1\} \setminus V^{\mathscr{C}}$.

(ii) To each leaf v, add a left and right successor from W.

Let $w_0, \ldots, w_{2r-1}$ be the sequence of these successors in the order from left to right.

(iii) For $i = 0$ to $r - 1$, do:

(a) Let v be the predecessor of w_{2i} and $Q' = l(w_0) \cup \cdots \cup l(w_{2i-1})$.

(b) Set $l_q(w_{2i}) = \Delta(l_q(v), a) \cap F \setminus Q'$ and $l_q(w_{2i+1}) = \Delta(l_q(v), a) \setminus (F \cup Q')$.

(c) Set $l_q(w_{2i}) = 2$ and $l_q(w_{2i+1}) = 0$.

(iv) Remove the leaf labels from the old leaves, that is, make l_q undefined for the predecessors of the new leaves. Mark every leaf which has label $\varnothing$. Recursively mark every vertex whose two successors are marked. Remove all marked vertices.

(v) Replace every nontrivial segment by its first vertex, and set its left labeling to

(a) 2 if one of the other vertices of the segment is labeled 1 or 2,

(b) 0 if each vertex of the segment is labeled 0, and

(c) 1 otherwise.

(vi) Replace the vertices from W by vertices from V_{new}.

(vii) Let R_0 be obtained from R by removing all vertices from $V^{\mathscr{C}} \setminus V^{\mathscr{C}'}$ from R and let R_1 be the sequence of all elements from $V^{\mathscr{C}'} \setminus V^{\mathscr{C}}$ according to the order $<$ on V. Then $R' = R_0 R_1$.

(viii) Let $h' \leq |R|$ be the maximal number such that $R(i) = R'(i)$ for all $i < h'$.

The full determinization construction is given in Figure 9. Summing up, we can state:

Theorem 2.14. (McNaughton-Safra-Piterman, [17, 105, 97]) Let $\mathscr{A}$ be a Büchi automaton with n states. Then $\mathscr{A}^{\det}$ is an equivalent deterministic parity automaton with $2^{\theta(n \log n)}$ states and $2n + 1$ different priorities.

Proof. The proof of the correctness of the construction described in Figure 9 is obvious from the previous analysis. The claim about the size of the resulting automaton can be established by simple counting arguments. Q.E.D.

Let $\mathscr{A}$ be a Büchi automaton. The deterministic parity automaton $\mathscr{A}^{\det}$ is defined by

$$\mathscr{A}^{\det} = (A, M, \mathscr{M}_I, \delta, \pi)$$

where

- M is the set of all Muller–Schupp trees over Q,
- $\mathscr{M}_I$ is the Muller–Schupp tree with just one vertex and leaf label Q_I,
- δ is such that $\delta(\mathscr{M}, a)$ is an a-successor of $\mathscr{M}$ (as defined above), and
- π is the priority function as defined for Muller–Schupp trees.

FIGURE 9. Determinization of a Büchi automaton

The previous theorem enables us to determine the expressive power of WS1S:

Corollary 2.15. There exists an effective procedure that given an S1S formula $\varphi = \varphi(V_0, \ldots, V_{m-1})$ produces a formula ψ such that $\mathscr{L}(\varphi) = \mathscr{L}^{\mathrm{w}}(\psi)$. In other words, every S1S formula is equivalent to a WS1S formula.

Sketch of proof. Given such a formula φ, one first uses Büchi's Theorem to construct a Büchi automaton $\mathscr{A}$ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathscr{A})$. In a second step, one converts $\mathscr{A}$ into an equivalent deterministic parity automaton $\mathscr{B}$, using the McNaughton–Safra–Piterman Theorem. The subsequent step is the crucial one. Assume $Q' = \{q_0, \ldots, q_{n-1}\}$ and, for every $u \in [2]_m^\omega$, let r_u be the (unique!) run of $\mathscr{B}$ on u. For every $i < n$, one constructs a formula $\psi_i = \psi_i(x)$ such that $u, j \models \psi_i(x)$ if and only if $r_u(j) = q_i$ for $u \in [2]_m^\omega$ and $j \in \omega$. These formulas can be built as in the proof of part 2 of Büchi's Theorem, except that one can restrict the sets X_i to elements $\leq j$, so weak quantification is enough. Finally, the formulas $\psi_i(x)$ are used to express acceptance. Q.E.D.

2.4 The Büchi–Landweber Theorem

The last problem remaining from the problems listed at the beginning of this section is realizability, also known as Church's problem [23, 24]. In our

context, it can be formalized more precisely as follows.

For letters $a \in [2]_m$ and $b \in [2]_n$, we define $a^\frown b \in [2]_{m+n}$ by $(a^\frown b)_{[i]} = a_{[i]}$ for $i < m$ and $a^\frown b_{[i]} = b_{[i-m]}$ for i with $m \leq i < m+n$. Similarly, when u and v are words of the same length over $[2]_n^\infty$ and $[2]_n^\infty$, respectively, then $u^\frown v$ is the word over $[2]_{m+n}$ with the same length defined by $(u^\frown v)(i) = u(i)^\frown v(i)$ for all $i < |u|$. Realizability can now be defined as follows: Given a formula

$$\varphi = \varphi(X_0, \ldots, X_{m-1}, Y_0, \ldots, Y_{n-1}),$$

determine whether there is a function $f\colon [2]_m^+ \to [2]_n$ such that $u^\frown v \models \varphi$ holds for every $u \in [2]_m^\omega$ and $v \in [2]_n^\omega$ defined by $v(i) = f(u[0,i])$.

Using the traditional terminology for decision problems, we say that φ is an instance of the realizability problem, f is a solution if it has the desired property, and φ is a positive instance if it has a solution.

Observe that the function f represents the device that produces the output in Figure 1: After the device has read the sequence $a_0 \ldots a_r$ of bit vectors (with m entries each), it outputs the bit vector $f(a_0 \ldots a_r)$ (with n entries).

In the above definition of realizability, we do not impose any bound on the complexity of f. In principle, we allow f to be a function which is not computable. From a practical point of view, this is not very satisfying. A more realistic question is to ask for a function f which can be realized by a finite-state machine, which is a tuple

$$\mathscr{M} = (A, B, S, s_I, \delta, \lambda)$$

where A is an input alphabet, B is an output alphabet, S is a finite set of states, $s_I \in S$ the initial state, $\delta\colon S \times A \to S$ the transition function, and $\lambda\colon S \to B$ the output function. To describe the function realized by $\mathscr{M}$ we first define $\delta^*\colon A^* \to S$ by setting $\delta(\varepsilon) = s_I$ and $\delta^*(ua) = \delta(\delta^*(u), a)$ for all $u \in A^*$ and $a \in A$. The function realized by $\mathscr{M}$, denoted $f_{\mathscr{M}}$, is defined by $f_{\mathscr{M}}(u) = \lambda(\delta^*(s_I, u))$ for every $u \in A^+$.

A solution f of an instance of the realizability problem is called a finite-state solution if it is realized by a finite-state machine.

Finite-state realizability is the variant of realizability where one is interested in determining whether a finite-state solution exists. We later see that there is no difference between realizability and finite-state realizability.

Several approaches have been developed to solving realizability; we follow a game-based approach. It consists of the following steps: We first show that realizability can be viewed as a game and that solving realizability means deciding who wins this game. We then show how the games associated with instances of the realizability problem can be reduced to finite games with a standard winning objective, namely the parity winning

condition. Finally, we use known results on finite games with parity winning conditions to prove the desired result.

2.4.1 Game-theoretic formulation

There is a natural way to view the realizability problem as a round-based game between two players, the environment and the (device) builder. In each round, the environment first provides the builder with an input, a vector $a \in [2]_m$, and then the builder replies with a vector $b \in [2]_n$, resulting in a combined vector $a^\frown b$. In this way, an infinite sequence of vectors is constructed, and the builder wins the play if this sequence satisfies the given S1S formula. Now, the builder has a winning strategy in this game if and only if the instance of the realizability problem we are interested in is solvable.

We make this more formal in what follows. A game is a tuple

$$\mathscr{G} = (P_0, P_1, p_I, M, \Omega)$$

where P_0 is the set of positions owned by Player 0, P_1 is the set of positions owned by Player 1 (and disjoint from P_0), $p_I \in P_0 \cup P_1$ is the initial position, $M \subseteq (P_0 \cup P_1) \times (P_0 \cup P_1)$ is the set of moves, and $\Omega \subseteq (P_0 \cup P_1)^\omega$ is the winning objective for Player 0. The union of P_0 and P_1 is the set of positions of the game and is denoted by P.

A play is simply a maximal sequence of positions which can be obtained by carrying out moves starting from the initial position, that is, it is a word $u \in P^+ \cup P^\omega$ such that $u(0) = p_I$, $(u(i), u(i+1)) \in M$ for every $i < |u|$, and if $|u| < \omega$, then there is no p such that $(u(*), p) \in M$. This can also be thought of as follows. Consider the directed graph (P, M), which is called the game graph. A play is simply a maximal path through the game graph (P, M) starting in p_I.

A play u is a win for Player 0 if $u \in \Omega \cup P^* P_1$, else it is a win for Player 1. In other words, if a player cannot move he or she loses early.

A strategy for Player α are instructions for Player α how to move in every possible situation. Formally, a strategy for Player α is a partial function $\sigma\colon P^* P_\alpha \to P$ which

(i) satisfies $(u(*), \sigma(u)) \in M$ for all $u \in \mathrm{dom}(\sigma)$ and

(ii) is defined for every $u \in P^* P_\alpha \cap p_I P^*$ satisfying $u(i+1) = \sigma(u[0,i])$ for all $i < |u| - 1$ where $u(i) \in P_\alpha$.

Observe that these conditions make sure that a strategy is defined when Player α moves according to it. A play u is consistent with a strategy σ if $u(i+1) = \sigma(u[0,i])$ for all i such that $u(i) \in P_\alpha$. A strategy σ is called a winning strategy for Player α if every play consistent with σ is a win for Player α. We then say that Player α wins the game.

Let $\varphi = \varphi(X_0, \ldots, X_{m-1}, Y_0, \ldots, Y_{n-1})$ be an S1S formula. The game $\mathscr{G}[\varphi]$ is defined by

$$\mathscr{G}[\varphi] = ([2]_m, \{p_I\} \cup [2]_n, p_I, M, \Omega)$$

where p_I is some initial position not contained in $[2]_m \cup [2]_n$ and

$$M = ([2]_m \times [2]_n) \cup ((\{p_I\} \cup [2]_n) \times [2]_m),$$
$$\Omega = \{p_I u_0 v_0 \ldots : (u_0 \frown v_0)(u_1 \frown v_1) \ldots \models \varphi\}.$$

FIGURE 10. Game for a realizability instance

The analogue of a finite-state solution is defined as follows. A strategy σ for Player α is finite-memory if there exists a finite set C, called memory, an element $m_I \in C$, the initial memory content, a function $\mu\colon C \times P \to C$, called update function, and a function $\xi\colon C \times P_\alpha \to P$ such that $\sigma(u) = \xi(\mu^*(u), u(*))$ for every $u \in \mathrm{dom}(\sigma)$, where μ^* is defined as δ^* above. That is, the moves of Player α depend on the current memory contents and the current position.

An even stronger condition than being finite-state is being memoryless. A strategy σ is memoryless if it is finite-state for a memory C which is a singleton set. As a consequence, if σ is memoryless, then $\sigma(up) = \sigma(u'p)$ for all $u, u' \in P^*$ with $up, u'p \in \mathrm{dom}(\sigma)$. So in this case, we can view a strategy as a partial function $P_\alpha \to P$. In fact, we use such functions to describe memoryless strategies.

We can now give the game-theoretic statement of the realizability problem. For an instance φ, consider the game $\mathscr{G}[\varphi]$ described in Figure 10.

Lemma 2.16. Let $\varphi = \varphi(X_0, \ldots, X_{m-1}, Y_0, \ldots, Y_{n-1})$ be an S1S formula. Then the following are equivalent:

(A) The instance φ of the realizability problem is solvable.

(B) Player 0 wins the game $\mathscr{G}[\varphi]$.

Moreover, φ is a positive instance of finite-state realizability if and only if Player 0 has a finite-memory winning strategy in $\mathscr{G}[\varphi]$.

Proof. For the implication from (A) to (B), let $f\colon [2]_m^+ \to [2]_n$ be the solution of an instance φ of the realizability problem. We define a par-

tial function $\sigma\colon p_I([2]_m[2]_n)^*[2]_m \to [2]_n$ by setting $\sigma(p_I a_0 b_1 \dots b_{r-1} a_r) = f(a_0 \dots a_r)$ where $a_i \in [2]_m$ for $i \leq r$ and $b_j \in [2]_n$ for $j < r$. It is easy to see that σ is a winning strategy for Player 0 in $\mathscr{G}[\varphi]$. Conversely, a winning strategy σ for Player 0 can easily be transformed into a solution of the instance φ of the realizability problem.

To prove the additional claim, one simply needs to observe that the transformations used in the first part of the proof convert a finite-state solution into a finite-memory strategy, and vice versa. The state set of the finite-state machine used to show that a solution to the realizability problem is finite-state can be used as memory in a proof that the winning strategy constructed above is finite-memory, and vice versa. Q.E.D.

In our definition of game, there is no restriction on the winning objective Ω, but since we are interested in winning objectives specified in S1S, we focus on parity winning conditions—remember that every S1S formula can be turned into a deterministic parity automaton. It will turn out that parity conditions are particularly apt to an algorithmic treatment while being reasonably powerful.

2.4.2 Reduction to finite parity games

A winning objective Ω of a game $\mathscr{G}$ is a parity condition if there is a natural number n and a function $\pi\colon P \to n$ such that $u \in \Omega$ iff $\mathrm{val}_\pi(u) \bmod 2 = 0$ for all $u \in P^\omega$. If this is the case, we replace Ω by π and speak of a parity game.

We next show that if Ω is a winning objective and $\mathscr{A}$ a deterministic parity automaton such that $\mathscr{L}(\mathscr{A}) = \Omega$, then we can "expand" a game $\mathscr{G}$ with winning objective Ω into a parity game, simply by running $\mathscr{A}$ in parallel with the moves of the players. The respective product construction is given in Figure 11.

Lemma 2.17. Let $\mathscr{G}$ be a finite game and $\mathscr{A}$ a deterministic parity automaton such that $\mathscr{L}(\mathscr{A}) = \Omega$. Then the following are equivalent:

(A) Player 0 wins $\mathscr{G}$.

(B) Player 0 wins $\mathscr{G} \times \mathscr{A}$.

Moreover, there exists a finite-memory winning strategy for Player 0 in $\mathscr{G}$ iff there exists such a strategy in $\mathscr{G} \times \mathscr{A}$.

Proof. The proof is straightforward. We transform a winning strategy for Player 0 in $\mathscr{G}$ into a winning strategy for Player 0 in $\mathscr{G} \times \mathscr{A}$ and vice versa.

First, we define u^δ for every $u \in P^*$ to be a word of the same length where the letters are determined by $u^\delta(i) = (u(i), \delta^*(q_I, u[0,i]))$ for every $i < |u|$.

Let $\mathscr{G}$ be a game and $\mathscr{A}$ a deterministic parity automaton with alphabet P such that $\mathscr{L}(\mathscr{A}) = \Omega$. The expansion of $\mathscr{G}$ by $\mathscr{A}$ is the game

$$\mathscr{G} \times \mathscr{A} = (P_0 \times Q, P_1 \times Q, (p_I, q_I), M', \pi')$$

where

$$M' = \{((p,q),(p',\delta(q,p'))) \colon q \in Q \wedge (p,p') \in \Delta\}$$

and $\pi'((p,q)) = \pi(q)$ for all $p \in P$ and $q \in Q$.

FIGURE 11. Product of a game with a deterministic parity automaton

Let $\sigma \colon P^*P_0 \to P$ be a winning strategy for Player 0 in $\mathscr{G}$. We transform this into $\sigma' \colon (P \times Q)^*(P_0 \times Q) \to P \times Q$ by letting $\sigma'(u^\delta) = \sigma(u)$ for every $u \in \mathrm{dom}(\sigma)$. It is easy to check that this defines a strategy and that this strategy is indeed winning.

Given a winning strategy $\sigma' \colon (P \times Q)^*(P_0 \times Q) \to P \times Q$, we define a winning strategy $\sigma \colon P^*P_0 \to P$ for Player 0 simply by forgetting the second component of the positions. That is, for every u such that $u^\delta \in \mathrm{dom}(\sigma')$ we set $\sigma(u) = \sigma'(u^\delta)$. Observe that this does not lead to any ambiguities, that is, σ is well-defined, because $\mathscr{A}$ is a deterministic automaton. It is easy to check that this defines a strategy and that this strategy is indeed winning.

If we have a finite-memory strategy σ for $\mathscr{G}$, say with memory C, we can use the same memory C and a modified update function to show that σ' as defined above is finite-state. Conversely, if we have a finite-memory strategy σ', say with memory C, we can use memory $Q \times C$ to show that σ as constructed above is finite-memory, too. Q.E.D.

Corollary 2.18. Let $\varphi = \varphi(X_0, \ldots, X_{m-1}, Y_0, \ldots, Y_{n-1})$ be an instance of the realizability problem for S1S and $\mathscr{A}$ a deterministic parity automaton recognizing $\mathscr{L}(\varphi)$. Then the following are equivalent:

(A) The instance φ of the realizability problem is solvable.

(B) Player 0 wins the game $\mathscr{G} \times \mathscr{A}$.

Moreover, if Player 0 has a finite-memory winning strategy in $\mathscr{G} \times \mathscr{A}$, then φ has a finite-state solution.

Using the fact that it can be determined effectively whether Player 0 wins a finite parity game (see Theorem 2.20 below), we obtain:

Theorem 2.19 (Büchi-Landweber, [19])**.** The realizability problem is decidable for S1S.

2.4.3 Background on games

In this section, we provide background on games, which we already used to solve Church's problem and which we need in various places.

Since plays of games may be infinite, it is not at all clear whether in a given game one of the two players has a winning strategy, that is, whether the game has a winner. When this is the case one says that the game is determined. It is said to be memoryless determined if there exists a memoryless winning strategy.

Theorem 2.20 (Emerson-Jutla-Mostowski, [40, 88])**.** Every parity game is memoryless determined.

That every parity game is determined follows immediately from a result by Martin [82].

For S1S realizability it is enough to know that the winner in a parity game can be effectively determined. In a later section, we need to know more about the computational complexity of this problem, in particular, we need to know how it depends on the number of priorities occurring in a game:

Theorem 2.21 (Jurdziński, [62])**.** Every parity game with n positions, m edges, and at most d different priorities in every strongly connected component of its game graph can be decided in time $O(n + mn^{\lfloor d/2 \rfloor})$ and an appropriate memoryless winning strategy can be computed within the same time bound.

2.5 Notes

Büchi's Theorem has been the blueprint for many theorems characterizing monadic second-order logic by automata. The most important theorem to mention is Rabin's Theorem [100], which extends Büchi's Theorem to the monadic theory of two successor functions and is the subject of the next section. Other early results, besides the Büchi–Elgot–Trakthenbrot theorem and Büchi's Theorem, are a result by Büchi [18] on ordinals and a result by Doner [31] (see also Thatcher and Wright [115]), which characterizes monadic second-order logic over finite trees in terms of automata and allows to prove that the weak monadic theory of two successor relations is decidable. Later results deal, for instance, with finite and infinite traces (certain partial orders) [119, 33], see also [30], pictures (matrices with letters as entries) [51], see also [50, 83], and weighted automata [32]. In some of these cases, the proofs are much harder than for S1S and Büchi automata.

When establishing a characterization of automata in terms of monadic second-order logic, proving part 2—the description of the behavior of an

automaton by a monadic second-order formula—is straightforward very often and leads to existential monadic second-order formulas, just as for S1S. The other direction—from full monadic second-order logic to automata—fails, however, for various automaton models because closure under complementation (negation) cannot be shown. In such cases, a partial result can sometimes nevertheless be obtained by showing that every existential monadic second-order formula can be translated into an automaton. This is, for instance, the case for pictures [51], see also [83].

Büchi's Theorem characterizes monadic second-order logic in terms of finite-state automata on infinite words. It is only natural to ask whether there are fragments of monadic second-order logics or other logics similar in expressive power to monadic second-order logic that can be characterized in a comparable fashion. We have already seen that the existential fragment of S1S has the same expressive power as S1S, but one can prove that first-order logic with ordering (and successor) or with successor only is strictly less expressive than S1S. The first of the two logics can be characterized just as in the case of finite words as defining exactly

(i) the star-free languages of infinite words,

(ii) the languages expressible in linear-time temporal logic, and

(iii) the languages of infinite words which are recognized by counter-free automata

(see [64, 116, 95, 132]), the second can be characterized as the weak version of locally threshold testability [117].

Ever since Büchi's seminal work automata on infinite words and formal languages of infinite words have been a major topic in research, motivated both from a mathematical and a computer science perspective. There have been many (successful) attempts to adapt the facts known from classical automata theory and the classical theory of formal languages to the setting with infinite words, for instance, regular expressions were extended to ω-regular expressions and the algebraic theory of regular languages was extended to an algebraic theory of ω-regular languages. But there are also new issues that arise for infinite words, which are essentially irrelevant for finite words. For example, the set of infinite words over a given alphabet can easily be turned into a topological space and it is interesting to study how complex languages are that can be recognized by finite-state automata.

One particularly interesting issue are the different types of acceptance conditions that are available for automata on infinite words. In our exposition, we work with Büchi and parity acceptance, but there are many more acceptance conditions which are suggested and widely used throughout the literature. The most prominent are: Streett [113], Rabin [100], and

Muller conditions [89]. An important question regarding all these different acceptance conditions is which expressive power they have, depending on whether they are used with deterministic or nondeterministic automata. It turns out that when used with nondeterministic automata all the aforementioned conditions are not more powerful than nondeterministic Büchi automata and when used with deterministic automata they are all as powerful as deterministic parity automata. In other words, each of the three conditions is just as good as the parity condition. Given McNaughton's Theorem, this is not very difficult to show. In almost all cases, asymptotically optimal conversions between the various conditions are known [105]. Recent improvements are due to Yan [131].

It is not only the type of acceptance condition that can be varied, but also the type of "mode". In this section, we have dealt with deterministic and nondeterministic automata. One can either look for

(i) modes in between or

(ii) modes beyond nondeterminism.

As examples for (i) we mention unambiguous automata [4], which are Büchi automata which admit at most one accepting run for each word, and prophetic automata [21], which are Büchi automata with the property that there is exactly one run on each word (besides partial runs that cannot be continued), be it accepting or not.

Examples for (ii) are alternating automata on infinite words, which are explained in detail in Section 4. Since they are, in principle, stronger than nondeterministic automata, they often allow for more succinct representations, which is why they have been studied extensively from a practical and complexity-theoretic point of view. Moreover, they can often be used to make automata-theoretic constructions more modular and transparent and help to classify classes of languages. For instance, the Kupferman–Vardi complementation construction for Büchi automata uses what are called weak alternating automata as an intermediate model of automaton, see also [121].

As can be seen from the Büchi–Landweber theorem, games of infinite duration are intimately connected with the theory of automata on infinite words. This becomes even more obvious as soon as alternating automata come into the picture, because they can be viewed as defining families of games in a uniform fashion. These games play a similar role in the theory of automata on infinite trees, as will be explained in the next section. Regardless of this, these games are interesting in their own right and there is an extensive literature on them. One of the major open problems is the computational complexity of finite parity games. The best upper bounds are that the problem is in UP $\cap$ co-UP, which is a result by Jurdziński [61], that it can be solved by subexponential algorithms, see, for instance, [63],

and polynomial time algorithms when the underlying game graphs belong to certain restricted classes of graphs, see, for instance, [8].

3 Monadic-second order logic of two successors

Büchi's Theorem is a blueprint for Rabin's result on monadic second-order logic of two successors (S2S). The formulas of that logic are built just as S1S formulas are built, except that there are two successor relations and not only one. More precisely, while in S1S the atomic formulas are of the form $x \in X$ and $\mathrm{suc}(x, y)$ only, in S2S the atomic formulas are of the form $x \in X$, $\mathrm{suc}_0(x, y)$, and $\mathrm{suc}_1(x, y)$, where $\mathrm{suc}_0(x, y)$ and $\mathrm{suc}_1(x, y)$ are read as "y is the left successor of x" and "y is the right successor of x", respectively. S2S formulas are interpreted in the full binary tree $\mathscr{T}_{\mathrm{bin}}$.

As a first simple example, we design a formula with one free set variable X which holds true if and only if the set assigned to X is finite. This can be expressed by saying that on every branch there is a vertex such that the subtree rooted at this vertex does not contain any element from X. This leads to:

$$\forall Y(\text{“}Y \text{ is a branch of the binary tree”} \rightarrow \exists y(y \in Y \wedge \forall z(y \leq z \rightarrow \neg z \in X))),$$

where $\leq$ is meant to denote the prefix order on the vertices of $\mathscr{T}_{\mathrm{bin}}$. That Y is a branch of $\mathscr{T}_{\mathrm{bin}}$ can easily be expressed as a conjunction of several simple conditions:

- Y is not empty, which can be stated as $\exists x(x \in Y)$,
- with each element of Y its predecessor (provided it exists) belongs to Y, which can be stated as $\forall x \forall y(y \in Y \wedge (\mathrm{suc}_0(x, y) \vee \mathrm{suc}_1(x, y)) \rightarrow x \in Y)$, and
- each element of Y has exactly one successor in Y, which can be stated as $\forall x \forall y \forall z(x \in Y \wedge \mathrm{suc}_0(x, y) \wedge \mathrm{suc}_1(x, z) \rightarrow (y \in Y \leftrightarrow z \notin Y))$.

To conclude the example, we define $x \leq y$ by stating that every successor-closed set containing x contains y as well:

$$\forall X(x \in X \wedge \forall z \forall z'(z \in X \wedge (\mathrm{suc}_0(z, z') \vee \mathrm{suc}_1(z, z')) \rightarrow z' \in X) \rightarrow y \in X).$$

Observe that we have a universally quantified set variable in this formula, whereas in Section 2 we use an existentially quantified set variable to define ordering for the natural numbers. In both situations, one can use either type of quantifier.

As a second example, we consider the property that on every branch there are only finitely many elements from X. This can be specified by:

$$\forall Y(\text{“}Y \text{ is a branch of the binary tree”} \rightarrow \exists y(y \in Y \wedge \forall z(x \leq z \wedge z \in Y \rightarrow \neg z \in X))),$$

using the same auxiliary formulas from above.

The most important question about S2S is whether satisfiability is decidable. A positive answer to this question implies decidability of the monadic second-order theory of the binary tree and a number of related theories as Rabin showed in his 1969 paper [100].

That satisfiability of an S2S formula is decidable can, in principle, be shown in the same way as the analogous statement for S1S: One first proves that every S2S formula can be translated into an equivalent automaton—this time a tree automaton—and then shows that emptiness for the automata involved is decidable. This is the approach that Rabin took in [100], and which we follow here, too.

3.1 Rabin's Theorem

In his original paper [100] Rabin used what we nowadays call Rabin tree automata to characterize S2S. We use the same model of tree automaton but with a simpler acceptance condition, the parity acceptance condition, which we also use in the context of S1S.

It is not clear right away how a tree automaton model should look like, but it turns out that it is reasonable to envision a tree automaton as follows. Starting in an initial state at the root of the tree the automaton splits up into two copies, one which proceeds at the left successor of the root and one which proceeds at the right successor of the root. The states which are assumed at these vertices are determined by the initial state and the label of the root. Then, following the same rules, the copy of the automaton residing in the left successor of the root splits up into two copies which proceed at the left successor of the left successor of the root and the right successor of the left successor of the root, and so on. In this way, every vertex of the tree gets assigned a state, and a tree is accepted if the state labeling of each branch satisfies the acceptance condition.

Formally, a parity tree automaton is a tuple

$$\mathscr{A} = (A, Q, q_I, \Delta, \pi),$$

where A, Q, and π are as with parity (word) automata (see Section 2.3.1), q_I is an initial state instead of a set of initial states, and Δ is a transition relation satisfying $\Delta \subseteq Q \times A \times Q \times Q$. Such an automaton runs on full A-labeled binary trees which are given implicitly. A run of $\mathscr{A}$ on a binary tree

$t\colon 2^* \to A$ is a binary tree $r\colon 2^* \to Q$ such that $(r(u), t(u), r(u0), r(u1)) \in \Delta$ for all $u \in 2^*$. It is accepting if for every infinite branch $u \in 2^\omega$ its labeling satisfies the parity condition, that is, if $\mathrm{val}_\pi(r(u(0))r(u(1))\dots) \bmod 2 = 0$.

As an example, consider the set L of all binary trees over $\{0,1\}$ with only finitely many vertices labeled 1 on each branch, which is very similar to the second property discussed above. It is straightforward to construct a parity tree automaton that recognizes L. The main idea is to use two states, q_0 and q_1, to indicate which label has just been read and to use the parity condition to check that on every path there are only finitely many vertices labeled q_1. In other words, we have $A = \{0,1\}$, $Q = \{q_I, q_0, q_1\}$, $\Delta = \{(q, a, q_a, q_a)\colon a \in A, q \in Q\}$, $\pi(q_I) = 0$, and $\pi(q_a) = a + 1$ for $a \in A$.

Rabin, in [100], proved a complete analogue of Büchi's theorem. We state Rabin's Theorem using the same notation as in the statement of Büchi's Theorem, which means, for instance, that we write $\mathscr{L}(\mathscr{A})$ for the set of all trees accepted by a parity tree automaton $\mathscr{A}$.

Theorem 3.1 (Rabin, [100])**.**

(i) There exists an effective procedure that given an S2S formula $\varphi = \varphi(V_0, \dots, V_{m-1})$ outputs a parity tree automaton $\mathscr{A}$ such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\varphi)$.

(ii) There exists an effective procedure that given a parity tree automaton $\mathscr{A}$ over an alphabet $[2]_m$ outputs a formula $\varphi = \varphi(V_0, \dots, V_{m-1})$ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathscr{A})$.

To prove part 2 one can follow the same strategy as with S1S: One simply constructs a formula that describes an accepting run of a given parity tree automaton. Proofs of part 2 of Theorem 3.1 can also be carried out as with S1S: One uses a simple induction on the structure of the formula. The induction base and all but one case to be considered in the inductive step are almost straightforward. The difficult step is—just as with Büchi automata—negation. One has to show that the complement of a tree language recognized by a parity tree automaton can be recognized by a parity tree automaton. This result, also known as Rabin's complementation lemma, can be proved in different ways. We present a proof which, in spirit, is very similar to what can be found in Büchi's [12] and Gurevich and Harrington's [54] work. At its heart, there is a game-theoretic description of acceptance (Section 3.2). The complementation construction itself has the determinization from Theorem 2.14 built in (Section 3.3).

3.2 The automaton-pathfinder game

Let $\mathscr{A}$ be a parity tree automaton as above and $t\colon 2^* \to A$ a binary tree. We consider a parity game where one can think of Player 0 as proving to

Let $\mathscr{A}$ be a parity tree automaton and $t\colon 2^* \to A$ a tree over the same alphabet. The automaton-pathfinder game for $\mathscr{A}$ and t is the parity game $\mathscr{G}[\mathscr{A}, t]$ defined by

$$\mathscr{G}[\mathscr{A}, t] = (2^* \times Q, 2^* \times \Delta, (\varepsilon, q_I), M_0 \cup M_1, \pi')$$

where

- for every word $u \in 2^*$, state $q \in Q$, and $(q, t(u), q_0, q_1) \in \Delta$, the move $((u, q), (u, (q, t(u), q_0, q_1)))$ belongs to M_0,
- for every word $u \in 2^*$, transition $(q, t(u), q_0, q_1) \in \Delta$, and $i < 2$, the move $((u, (q, t(u), q_0, q_1)), (ui, q_i))$ belongs to M_1, and
- $\pi'((u, q)) = \pi(q)$ for all $u \in 2^*$ and $q \in Q$.

FIGURE 12. Automaton-pathfinder game

Player 1 that t is accepted by $\mathscr{A}$, as follows. The game starts at the root of the tree and Player 0 suggests a transition which works at the root of the tree, which means it must start with the initial state and it must show the symbol the root is labeled with. Then Player 1 chooses the left or right successor of the root, say she chooses the left successor. Now it's Player 0's turn again. He must choose a transition which works for the left successor, which means it must start with the state chosen for the left successor in the transition chosen in the previous round and it must show the symbol the left successor is labeled with. Then Player 1 chooses one of the two successors, and so on. As the play proceeds, a sequence of transitions is constructed. Player 0 wins this play when the respective sequence of the source states of the transitions satisfies the parity condition.

The precise definition of the parity game is given in Figure 12. Observe that for convenience the priority function is only partially defined. This does not cause any problems since there is an infinite number of vertices with priorities assigned to them on every infinite path through the game graph.

Lemma 3.2 (Gurevich-Harrington, [54])**.** Let $\mathscr{A}$ be a parity tree automaton and $t\colon 2^* \to A$ a tree over the same alphabet. Then the following are equivalent:

(A) $\mathscr{A}$ accepts t.

(B) Player 0 wins $\mathscr{G}[\mathscr{A}, t]$.

Proof. For the implication from (A) to (B), we show how to convert an accepting run $r\colon 2^* \to Q$ of $\mathscr{A}$ on t into a winning strategy for Player 0 in $\mathscr{G}[\mathscr{A}, t]$. A strategy σ for Player 0 is defined on words of the form

$$u = (\varepsilon, q_0)(\varepsilon, \tau_0)(a_0, q_1)(a_0, \tau_1)(a_0 a_1, q_2) \dots (a_0 \dots a_{n-1}, q_n)$$

with $q_0 = q_I$, $q_i \in Q$ for $i \leq n$, $\tau_i \in \Delta$, and $a_i \in \{0, 1\}$ for $i < n$. For such a word u, we set $v_u = a_0 \dots a_{n-1}$. After the explanations given above on how one should think of the game, it should be clear that we set $\sigma(u) = (u, (q_n, a, q^0, q^1))$ with $q^i = r(v_u i)$ for $i < 2$. It is easy to check that this defines a winning strategy, because every play conform with σ corresponds to a branch of the run r.

Conversely, assume σ is a winning strategy for Player 0 in the above game. Then an accepting run r can be defined as follows. For every partial play u as above which is conform with σ, we set $r(v_u) = q_n$. It is straightforward to check that this defines an accepting run, because every path in r corresponds to a play of $\mathscr{G}[\mathscr{A}, t]$ conform with σ. Q.E.D.

There is a similar parity game—the emptiness game—which describes whether a given parity tree automaton accepts some tree. In this game, when Player 0 chooses a transition, he does not need to take into account any labeling; he simply needs to make sure that the transition is consistent with the previously chosen transition. The full game is described in Figure 13.

With a proof similar to the one of Lemma 3.2, one can show:

Lemma 3.3. Let $\mathscr{A}$ be a parity tree automaton. Then $\mathscr{L}(\mathscr{A}) \neq \varnothing$ if and only if Player 0 wins $\mathscr{G}_{\varnothing}[\mathscr{A}]$.

Taking Theorem 2.21 into account, we obtain:

Corollary 3.4 (Rabin, [100])**.** The emptiness problem for parity tree automata is decidable.

Rabin proved, in some sense, a stronger result, because he used tree automata with Rabin acceptance condition. As a further consequence of Lemma 3.3, taking Rabin's Theorem into account, we note:

Corollary 3.5 (Rabin, [100])**.** Satisfiability is decidable for S2S.

Let $\mathscr{A}$ be a parity tree automaton. The emptiness game $\mathscr{G}_\varnothing[\mathscr{A}]$ is defined by

$$\mathscr{G}_\varnothing[\mathscr{A}] = (Q, \Delta, q_I, M_0 \cup M_1, \pi)$$

where

- for $q \in Q$ and $(q, a, q_0, q_1) \in \Delta$, the move $(q, (q, a, q_0, q_1))$ belongs to M_0,
- for every $(q, a, q_0, q_1) \in \Delta$ and $i < 2$, the move $((q, a, q_0, q_1), q_i)$ belongs to M_1.

FIGURE 13. Emptiness game for a parity tree automaton

3.3 Complementation of parity tree automata

We can finally turn to the question of how to arrive at a parity tree automaton for the complement of a set of trees accepted by a given parity tree automaton. We are given a parity tree automaton $\mathscr{A}$ and we want to construct a parity tree automaton which recognizes $\mathscr{L}(\mathscr{A})^C$, where for each tree language L over some alphabet A we write L^C for the set of all trees over A which do not belong to L.

We describe the entire construction as a composition of several simpler constructions. More precisely, we first show that for every tree in the complement there exists a tree over an enhanced alphabet which witnesses its membership to the complement. The second step is to prove that the set of these witnesses can be recognized by a universal parity tree automaton. The third step consists in showing that universal parity tree automaton can be converted into (ordinary nondeterministic) parity tree automata, and the final step shows how to reduce the enhanced alphabet to the real one.

The first key step is to combine the automaton-pathfinder game with memoryless determinacy. To this end, we encode memoryless (winning) strategies for the pathfinder in trees. Observe that a memoryless strategy for the pathfinder in $\mathscr{G}[\mathscr{A}, t]$ for some automaton $\mathscr{A}$ and some tree t is simply a (partial) function $\sigma\colon 2^* \times \Delta \to 2^* \times Q$. Since, by construction of $\mathscr{G}[\mathscr{A}, t]$, we always have $\sigma(u, (q, a, q_0, q_1)) = (ui, q_i)$ for some $i < 2$, we can view such a function as a function $2^* \times \Delta \to 2$, which, in turn, can be viewed as a function $2^* \to 2^\Delta$. The latter is simply a 2^Δ-labeled tree. When we further encode the given tree t in that tree, we arrive at the following

notion of complement witness.

Let $\mathscr{A}$ be a parity tree automaton and $t' \colon 2^* \to A \times 2^\Delta$ a tree. For simplicity, we write $t'(u)$ as (a_u, f_u) for every $u \in 2^*$. The tree t' is a complement witness if for every branch $u \in 2^\omega$ the following holds. If $\tau_0\tau_1 \dots \in \Delta^\omega$ with $\tau_i = (q_i, a_{u[0,i)}, q_i^0, q_i^1)$ is such that $q_0 = q_I$ and $q_{i+1} = q_i^b$ where $b = f_{u[0,i)}(\tau_i)$ for every i, then $\mathrm{val}_\pi(q_0q_1 \dots) \bmod 2 = 1$, that is, $q_0q_1 \dots$ is not accepting with respect to π.

After the explanation given above, Theorem 2.20 now yields the lemma below, where we use the following notation. Given a tree $t' \colon 2^* \to A \times B$ for alphabet A and B, we write $\mathrm{pr}_0(t')$ for the tree defined by $\mathrm{pr}_0(t')(u) = \mathrm{pr}_0(t'(u))$ for every $u \in 2^*$, that is, we simply forget the second component of every label.

Lemma 3.6. Let $\mathscr{A}$ be a parity tree automaton and $t \colon 2^* \to A$ a tree over the same alphabet. Then the following are equivalent:

(A) $t \in \mathscr{L}(\mathscr{A})^C$.

(B) There is a complement witness t' for $\mathscr{A}$ such that $\mathrm{pr}_0(t') = t$. Q.E.D.

Using more notation, we can state the above lemma very concisely. First, we extend projection to tree languages, that is, given a tree language L over some alphabet $A \times B$, we write $\mathrm{pr}_0(L)$ for $\{\mathrm{pr}_0(t) \colon t \in L\}$. Second, given a parity tree automaton $\mathscr{A}$, we write $\mathscr{C}(\mathscr{A})$ for the set of all complement witnesses for $\mathscr{A}$. Then Lemma 3.6 simply states:

Remark 3.7. For every parity tree automaton $\mathscr{A}$,

$$\mathscr{L}(\mathscr{A})^{\mathsf{C}} = \mathrm{pr}_0(\mathscr{C}(\mathscr{A})).$$

So, clearly, once we have a parity tree automaton for $\mathscr{C}(\mathscr{A})$, we also have a parity tree automaton for $\mathscr{L}(\mathscr{A})^{\mathsf{C}}$, because we only need to omit the second component from the letters in the transition function to obtain the desired automaton.

It is not straightforward to find a parity tree automaton that recognizes $\mathscr{C}(\mathscr{A})$; it is much easier to show that $\mathscr{C}(\mathscr{A})$ is recognized by a universal parity tree automaton. Such an automaton is a tuple

$$\mathscr{A} = (A, Q, q_I, \Delta, \pi)$$

where A, Q, q_I, and π are as with parity tree automata and $\Delta \subseteq Q \times A \times 2 \times Q$. Let $t \colon 2^* \to A$ be a tree over A. A word $r \in Q^\omega$ is said to be a run for branch $u \in 2^\omega$ if $(r(i), t(u[0,i)), u(i), r(i+1)) \in \Delta$ for every i and $r(0) = q_I$. A tree is accepted if every $r \in Q^\omega$ which is a run for some branch satisfies the parity acceptance condition.

Let $\mathscr{A}$ be a parity tree automaton. The universal parity tree automaton $\mathscr{A}^{\mathrm{cw}}$ is defined by

$$\mathscr{A}^{\mathrm{cw}} = (A \times 2^{\Delta}, Q, q_I, \Delta', \pi + 1)$$

where $(q, (a, f), d, q') \in \Delta'$ if there exists $\tau = (q, a, q_0, q_1) \in \Delta$ such that $f(\tau) = d$ and $q_d = q'$, and where $\pi + 1$ stands for the priority function π' defined by $\pi'(q) = \pi(q) + 1$.

FIGURE 14. Universal parity tree automaton for complement witnesses

We can now rephrase Lemma 3.6 in terms of the new automaton model. We can express the complement of a tree language recognized by a parity tree automaton as the projection of a tree language recognized by a universal parity tree automaton. The latter is defined in Figure 14. Observe that the runs for the branches in this automaton correspond to the words $\tau_0\tau_1\ldots$ in the definition of complement witness.
We immediately obtain:

Remark 3.8. For every parity tree automaton $\mathscr{A}$,

$$\mathscr{C}(\mathscr{A}) = \mathscr{L}(\mathscr{A}^{\mathrm{cw}}).$$

To complete the description of the complementation procedure, we need to explain how a universal parity tree automaton can be converted into a parity tree automaton. One option for such a construction is to use McNaughton's Theorem, namely that every nondeterministic Büchi automaton can be turned into a deterministic parity automaton. The idea is that the tree automaton follows all runs of a given branch at the same time by running a deterministic word automaton in parallel.

Let Q be a finite set of states and $\pi\colon Q \to \omega$ a priority function. Let $\mathscr{Q}$ be the alphabet consisting of all binary relations over Q. Then every word $u \in \mathscr{Q}^{\omega}$ generates a set of infinite words $v \in Q^{\omega}$, denoted $\langle u \rangle$, defined by

$$\langle u \rangle = \{v \in Q^{\omega} \colon \forall i((v(i), v(i+1)) \in u(i))\},$$

and called the set of paths through u, because one can think of $\langle u \rangle$ as the set of all infinite paths through the graph which is obtained by "collating" $u(0), u(1), \ldots$. We are interested in a deterministic parity automaton $\mathscr{A}[Q, \pi]$ which checks that all paths through a given u satisfy the given parity condition, that is, which has the following property. For every $u \in \mathscr{Q}^{\omega}$,

$$u \in \mathscr{L}(\mathscr{A}[Q, \pi]) \qquad \text{iff} \qquad \forall v(v \in \langle u \rangle \to \mathrm{val}_{\pi}(v) \bmod 2 = 0). \tag{1.1}$$

Let Q be a finite set of states and $\pi\colon Q \to \omega$ a priority function. Consider the parity word automaton

$$\mathscr{B} = (\mathscr{Q}, Q, Q_I, \Delta, \pi + 1)$$

where $\Delta = \{(q, R, q')\colon (q, q') \in R\}$. Let $\mathscr{C}$ be an equivalent Büchi automaton (Figure 7) and $\mathscr{D}$ a deterministic parity automaton equivalent to $\mathscr{C}$ (Figure 9). The automaton $\mathscr{A}[Q, \pi]$ is defined by

$$\mathscr{A}[Q, \pi] = (\mathscr{Q}, Q^{\mathscr{D}}, q_I^{\mathscr{D}}, \delta^{\mathscr{D}}, \pi + 1).$$

FIGURE 15. Generic automaton for state set and priority function

Using Theorem 2.14, such an automaton, which we call a generic automaton for Q and π, can easily be constructed, as can be seen from Figure 15. Observe that, by construction,

$$u \in \mathscr{L}(\mathscr{C}) \qquad \text{iff} \qquad \exists v(v \in \langle u \rangle \wedge \mathrm{val}_\pi(v) \bmod 2 = 1),$$

for every $u \in \mathscr{Q}^\omega$. We conclude:

Remark 3.9. Let Q be a finite state set and $\pi\colon Q \to \omega$ a priority function. Then 1.1 holds for every $u \in \mathscr{Q}^\omega$.

Given the generic automaton, it is now easy to convert universal tree automata into nondeterministic ones: One only needs to run the generic automaton on all paths. This is explained in detail in Figure 16.

Lemma 3.10. Let $\mathscr{A}$ be a universal parity tree automaton. Then $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\mathscr{A}^{\mathrm{nd}})$.

Proof. For convenience, we write $\mathscr{B}$ for $\mathscr{A}[Q^{\mathscr{A}}, \pi^{\mathscr{A}}]$.

First observe that for every $t\colon 2^* \to A$ there is exactly one run of $\mathscr{A}^{\mathrm{nd}}$ on t. This is because Δ is such that for every $s \in S$ and $a \in A$, there is exactly one transition in Δ of the form (s, a, s_0, s_1). For a given t, let r_t denote this run. So in order to determine whether a tree is accepted by $\mathscr{A}^{\mathrm{nd}}$, we only need to determine whether r_t is accepting. To this end, we consider a branch $w \in 2^\omega$ of this tree.

By construction of $\mathscr{A}^{\mathrm{nd}}$, the labeling of w in r_t is the run of $\mathscr{B}$ on $u = R_0^u R_1^u \ldots$ where $R_i^u = R_{t(w[0,i)),w(i)}$. So $\langle u \rangle$ is the set of runs of $\mathscr{A}$

Let $\mathscr{A}$ be a universal parity tree automaton and assume that the generic automaton for $Q^{\mathscr{A}}$ and $\pi^{\mathscr{A}}$ is given as $\mathscr{A}[Q^{\mathscr{A}}, \pi^{\mathscr{A}}] = (\mathscr{Q}^{\mathscr{A}}, S, s_I, \delta, \pi)$. The parity tree automaton $\mathscr{A}^{\mathrm{nd}}$ is defined by

$$\mathscr{A}^{\mathrm{nd}} = (A, S, s_I, \Delta, \pi)$$

where for every $a \in A$ and $s \in S$,

$$\tau_{s,a} = (s, a, \delta(s, R_{a,0}), \delta(s, R_{a,1}))$$

with $R_{a,d} = \{(q, q') \colon (q, a, d, q') \in \Delta^{\mathscr{A}}\}$ for $d < 2$ and

$$\Delta = \{\tau_{s,a} \colon a \in A \wedge s \in S\}.$$

FIGURE 16. From universal to nondeterministic parity tree automata

on branch w. In view of Remark 3.9, this implies that w is accepting as a branch of r_t if and only if all runs of $\mathscr{A}$ on w are accepting. From this, the claim of the lemma follows immediately. Q.E.D.

This was also the last missing piece in the construction from a given parity tree automaton to a parity tree automaton for its complement:

Lemma 3.11 (Rabin, [100])**.** There is an effective procedure that turns a given parity tree automaton $\mathscr{A}$ into a parity tree automaton $\mathscr{A}^C$ that recognizes the complement of the language recognized by $\mathscr{A}$. Q.E.D.

3.4 Notes

Rabin's Theorem is important from a mathematical (logical) point of view because it is a very strong decidability result and can as such be used to show the decidability of many theories, see, for instance, Rabin's original paper [100] and the book [15]. A very specific question to ask is how one can prove that the monadic second-order (or first-order) theory of a certain structure is decidable using the fact that it is decidable for the binary tree. There is a wide spectrum of techniques that have been developed to this end and are explained in detail in [9], see also [22].

It may seem that the results proved for S1S and automata on infinite words extend to S2S and automata on infinite trees in a straightforward fashion. This is true in many respects, but there are important differences.

Most importantly, it is neither true that every tree language recognized by a parity tree automaton can be recognized by a Büchi tree automaton nor is it true that WS2S and S2S are equally expressive. There is, however, an interesting connection between Büchi tree automata and WS2S: a set of trees is definable in WS2S if and only if it is recognized by a Büchi tree automaton and its complement is so, too, which was proved by Rabin [101]. Moreover, being definable in WS2S is equivalent to being recognized by a weak alternating tree automaton [73]. It is true though that every S2S formula is equivalent to an existential S2S formula. Also note that the second formula given as example at the beginning of this section is one which cannot be recognized by a Büchi tree automaton, let alone specified in WS2S. Another noticeable difference between automata on infinite words and automata on infinite trees is that unambiguous tree automata are weaker than nondeterministic ones, which is a result due to Niwiński and Walukiewicz [94]. Its proof was recently simplified considerably by Carayol and Löding [20].

The most complicated automata-theoretic building block of our proof of Rabin's theorem is McNaughton's Theorem, the determinization of word automata. It is not clear to which extent McNaughton's Theorem is necessary for the proof of Rabin's Theorem. The proof presented here is based on a translation in the sense that for every S2S formula φ we construct an automaton $\mathscr{A}$ such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\varphi)$ and it makes full use of a determinization construction. There are other proofs, such as the one by Kupferman and Vardi [75], which do not rely on the entire construction but only on the fact that there are determinization constructions with a certain bound on the number of states. These constructions, however, yield a slightly weaker result in the sense that they only reduce S2S satisfiability to tree automaton emptiness. In the proof presented here, determinization is used to turn a universal automaton into a nondeterministic one, which could be called a de-universalization construction. It would be interesting to see if one can also go in the reverse direction, that is, whether there is a determinization construction which can be built on a de-universalization construction.

At the end of the previous section, we mentioned that topological questions are interesting in the context of infinite words and automata on infinite words. This is even more true for infinite trees, see [5].

4 Linear-time temporal logic

Although originally introduced in this context, S1S and WS1S have only very rarely been used to specify properties of (finite-state) devices (see [56] for a noticeable exception). For S2S, this is even more true; it has almost always been used to obtain decidability for logical theories as pointed out

in Section 3. But the ever-increasing number of real computational devices and large-scale production lines of such devices has called for appropriate specification logics. In this section, we consider a logic that was introduced in this regard and show how it can be dealt with using automata theory, in particular, we show how specifically tailored automata can be used to obtain optimal upper bounds for problems such as satisfiability, conformance—in this context called model checking—, and realizability.

4.1 LTL and S1S

Linear-time temporal logic (LTL) is a modal logic designed to specify temporal relations between events occurring over time, designed by Kamp [64] to formally describe temporal relationships expressible in natural language and introduced into computer science by Pnueli [98] (see also the work by Burstall [13] and Kröger [69]) as an appropriate specification language for systems with nonterminating computations. Nowadays, LTL is widely spread and used in practice.

From a syntactic point of view LTL is propositional logic augmented by temporal operators. LTL formulas are built from tt and propositional variables using negation ($\neg$), disjunction ($\vee$), and the binary temporal operator XU called "strict until" and used in infix notation. For instance, when p is a propositional variable, then $\neg p \wedge \mathsf{tt}\,\mathsf{XU}\,p$ is an LTL formula. When P is a finite set of propositional variables and φ an LTL formula with propositional variables from P, then φ is called a formula over P.

LTL formulas are typically interpreted in infinite words, more precisely, given a finite set P of propositional variables, an LTL formula φ over P, a word $u \in (2^P)^\omega$, and $i \geq 0$, it is defined what it means that φ holds in u at position i, denoted $u, i \models \varphi$:

- $u, i \models \mathsf{tt}$,
- $u, i \models p$ if $p \in u(i)$, for every $p \in P$,
- $u, i \models \neg\varphi$ if $u, i \not\models \varphi$, for every LTL formula φ over P,
- $u, i \models \varphi \vee \psi$ if $u, i \models \varphi$ or $u, i \models \psi$, for LTL formulas φ and ψ over P, and
- $u, i \models \varphi\,\mathsf{XU}\,\psi$ if there exists $j > i$ such that $u, j \models \psi$ and $u, i' \models \varphi$ for all i' such that $i < i' < j$.

So $\varphi\,\mathsf{XU}\,\psi$ means that the formula φ holds true in the future until a point is reached where ψ holds true. For a word u as above and an LTL formula φ we say that φ holds in u, denoted $u \models \varphi$, if $u, 0 \models \varphi$. The language defined by φ is $\mathscr{L}(\varphi) = \{u \in (2^P)^\omega \colon u \models \varphi\}$, where, for convenience, we do not refer to P in the notation.

Clearly, there are many more basic temporal relations than just "until". So, often, other temporal operators are used:

- "next" is denoted X and defined by $\mathsf{X}\varphi = \neg\mathsf{tt}\,\mathsf{XU}\,\varphi$,
- "sometime in the future" is denoted XF and defined by $\mathsf{XF}\varphi = \mathsf{tt}\,\mathsf{XU}\,\varphi$, and
- "always in the future" is denoted XG and defined by $\mathsf{XG}\varphi = \neg\mathsf{XF}\neg\varphi$.

In many situations, it is convenient to include the current point in time, which leads to defining F by $\mathsf{F}\varphi = \varphi \vee \mathsf{XF}\varphi$ and, similarly, G by $\mathsf{G}\varphi = \neg\mathsf{F}\neg\varphi$ as well as U by $\varphi\,\mathsf{U}\,\psi = \psi \vee (\varphi \wedge \varphi\,\mathsf{XU}\,\psi)$.

It is remarkable that Kamp in his 1968 thesis [64] proved that every temporal relation expressible in natural (English) language can be expressed in linear-time temporal logic as defined above. As a yardstick for what is expressible in natural language he used first-order logic, considering formula with one free variable. To be precise, to obtain his result Kamp also had to add a past version of until, called since. That until by itself is enough to express everything expressible in first-order logic when only sentences are considered was proved by Gabbay, Pnueli, Shelah, and Stavi [47].

A typical LTL formula is

$$\mathsf{G}(p_r \to \mathsf{XF}p_a)$$

which expresses that for every occurrence of p_r there is a later occurrence of p_a, or, simply, every "request" is followed by an "acknowledge".

Another, more complicated, example is a formula expressing that competing requests are served in order. We assume that r_0 and r_1 are propositional variables indicating the occurrence of requests and a_0 and a_1 are matching propositional variables indicating the occurrence of acknowledgments. We want to specify that whenever an r_0 request occurs while no r_1 request is pending, then a_1 does not occur before the next occurrence of a_0.

We first specify that starting from an r_0 request there is an a_1 acknowledgment before an a_0 acknowledgment:

$$\alpha = r_0 \wedge (\neg a_0\,\mathsf{XU}\,(a_1 \wedge \neg a_0)).$$

Next, we observe that there are two different types of situations where an r_0 request can occur while no r_1 request is pending. The first type of situation is when there has been no r_1 request before the r_0 request in question. The second type is when a_1 occurred before the r_0 request in question and in between there has been no r_1 request. For each type of situation, we have a separate disjunct in our formula:

$$\neg(\neg r_1\,\mathsf{U}\,(\neg r_1 \wedge \alpha)) \vee \neg\mathsf{F}(a_1 \wedge \neg r_1\,\mathsf{U}\,(\neg r_1 \wedge \alpha)).$$

Clearly, in the context of LTL all the algorithmic problems discussed for S1S—satisfiability, conformance (model checking), and realizability—can be discussed. For instance, we can ask whether a given formula φ over P is satisfiable in the sense that there exists a word $u \in (2^P)^\omega$ such that $u \models \varphi$ or, given φ and a finite-state automaton $\mathscr{D}$ over 2^P, we can ask whether $u \models \varphi$ for all $u \in \mathscr{L}(\mathscr{D})$.

We can show in just one step that all these problems are decidable, namely by showing that every LTL formula is equivalent to an S1S formula; the results from Section 2 then apply. Unfortunately, the decision procedures that one obtains in this way have a nonelementary complexity. We can do better by using specifically tailored automata-theoretic constructions. We first present, however, the translation into S1S and then only turn to better decision procedures.

We start by defining the notion of equivalence we use to express the correctness of our translation. Let $P = \{p_0, \ldots, p_{r-1}\}$ be a set of propositional variables. Rather than interpreting LTL formulas over P in words over 2^P, we interpret them in words over $[2]_r$, where we think of every letter $a \in 2^P$ as the letter $b \in [2]_r$ with $b_{[j]} = 1$ iff $p_j \in a$ for every $j < r$. We say that an S1S formula $\psi = \psi(V_0, \ldots, V_{r-1})$ is equivalent to an LTL formula φ over P if for every $u \in [2]_r^\omega$ the following holds: $u \models \varphi$ iff $u \models \psi$.

In the proposition below, we make a stronger statement, and this involves the notion of global equivalence, which is explained next. Given a word $u \in [2]_r^\omega$, a position i, and an S1S formula $\psi = \psi(V_0, \ldots, V_{r-1}, x)$ where x is a first-order variable, we write $u, i \models \psi$ if ψ holds true when the set variables are assigned values according to u and x is assigned i. We say that ψ is globally equivalent to an LTL formula φ over P if the following holds: $u, i \models \varphi$ iff $u, i \models \psi$ for every $u \in [2]_r^\omega$ and every i.

Proposition 4.1. Let $P = \{p_0, \ldots, p_{r-1}\}$ be a finite set of propositional variables and x a first-order variable. For every LTL formula φ over P a globally equivalent S1S formula $\tilde{\varphi} = \varphi(V_0, \ldots, V_{r-1}, x)$ can be constructed.

Observe that $\exists x(\forall y \neg \mathrm{suc}(y, x) \wedge \tilde{\varphi})$ is equivalent to φ.

Proof. A proof can be carried out by a straightforward induction on the structure of φ. When $\varphi = \mathsf{tt}$, we choose $\tilde{\varphi} = (x = x)$, and when $\varphi = p_j$, we take $\tilde{\varphi} = x \in V_j$.

In the inductive step, we distinguish various cases. When $\varphi = \neg\psi$, we can choose $\tilde{\varphi} = \neg\tilde{\psi}$. Similarly, when $\varphi = \psi \vee \chi$, we can choose $\tilde{\varphi} = \tilde{\psi} \vee \tilde{\chi}$. Finally, assume $\varphi = \psi \,\mathsf{XU}\, \chi$. Then we choose

$$\tilde{\varphi} = \exists z(x < z \wedge \tilde{\chi}(V_0, \ldots, V_{r-1}, z) \wedge \forall y(x < y < z \rightarrow \tilde{\psi}(V_0, \ldots, V_{r-1}, y))),$$

which simply reflects the semantics of XU. Q.E.D.

Observe that the above proof even shows that every formula is equivalent to a first-order formula (without set quantification but with ordering), and a slightly more careful proof would show that three first-order variables are sufficient [58]. Kamp's seminal result [64] is the converse of the above proposition when first-order logic with ordering is considered instead of S1S.

As a consequence of Proposition 4.1, we can state:

Corollary 4.2. LTL satisfiability, model-checking, and realizability are decidable.

This result is not very satisfying, because in view of [112, 111] the decision procedures obtained in this way have nonelementary complexity. As it turns out, it is much better to translate LTL directly into Büchi automata and carry out the same constructions we have seen for S1S all over again. The key is a good translation from LTL into Büchi automata.

4.2 From LTL to Büchi automata

Vardi and Wolper [124, 126] were the first to describe and advocate a separate translation from LTL into Büchi automata, resulting in essentially optimal bounds for the problems dealt with in Section 2. These bounds were originally achieved by Sistla and Clarke [108, 109], for satisfiability and model checking, and by Pnueli and Rosner [99], for realizability.

There are several ways of translating LTL into Büchi automata. We present two translations, a classical and a modern translation: the first one goes from an LTL formula via a generalized Büchi automaton to an ordinary Büchi automaton, while the second one goes via very weak alternating automata.

Both of the constructions we are going to present are based on formulas in positive normal form, which we define next. The operator "release", denoted XR, is defined by $\varphi \mathsf{XR} \psi = \neg(\neg\varphi \mathsf{XU} \neg\psi)$. In a certain sense, $\varphi \mathsf{XR} \psi$ expresses that the requirement of ψ to hold is released by the occurrence of φ. LTL formulas in positive normal form are built starting from tt, ff, p, and $\neg p$ using $\vee$, $\wedge$, XU, and XR, that is, negations are only allowed to occur right in front of propositional variables.

The following identities show that every LTL formula can be transformed into an equivalent LTL formula in positive normal form which is not longer than the given one (not counting negation symbols).

Lemma 4.3. For LTL formulas φ and ψ over a finite set P of propositional

variables, $u \in (2^P)^\omega$, and $i \geq 0$, the following holds:

$$\begin{aligned}
u, i &\models \neg \mathsf{tt} && \text{iff} && u, i \models \mathsf{ff}, \\
u, i &\models \neg\neg\varphi && \text{iff} && u, i \models \varphi, \\
u, i &\models \neg(\varphi \vee \psi) && \text{iff} && u, i \models \neg\varphi \wedge \neg\psi, \\
u, i &\models \neg(\varphi \,\mathsf{XU}\, \psi) && \text{iff} && u, i \models \neg\varphi \,\mathsf{XR}\, \neg\psi.
\end{aligned}$$

Proof. A proof can be carried out in a straightforward fashion, using the definition of the semantics of LTL. Q.E.D.

As mentioned above, the other ingredient for our translation are generalized Büchi automata, introduced in [49]. Such an automaton is a tuple

$$\mathscr{A} = (A, Q, Q_I, \Delta, \mathscr{F})$$

where the first four components are as with ordinary Büchi automata, the only difference is in the last component: $\mathscr{F}$ is a set of subsets of Q, each called an acceptance set of $\mathscr{A}$. A run r is accepting if for every acceptance set $F \in \mathscr{F}$ there exist infinitely many i such that $r(i) \in F$. So generalized Büchi automata can express conjunctions of acceptance conditions in a simple way.

The essential idea for constructing a generalized Büchi automaton equivalent to a given LTL formula is as follows. As the automaton reads a given word it guesses which subformulas are true. At the same time it verifies its guesses. This is straightforward for almost all types of subformulas, for instance, when the automaton guesses that $\neg p$ is true, it simply needs to check that $p \notin a$ if a is the current symbol read. The only subformulas that are difficult to handle are XU-subformulas, that is, subformulas of the form $\psi \,\mathsf{XU}\, \chi$. Checking that such a subformula is true cannot be done directly or in the next position in general because the "satisfaction point" for an XU-formula—the position where χ becomes true—can be in the far future. Of course, by keeping $\psi \,\mathsf{XU}\, \chi$ in the state the automaton can remember the obligation to eventually reach a satisfaction point, but the acceptance condition is the only feature of the automaton which can be used to really check that reaching the satisfaction point is not deferred forever.

The complete construction is described in Figure 17; it uses $\mathrm{sub}(\varphi)$ to denote the set of all subformulas of a formula φ including φ itself. Note that for every XU-subformula $\psi \,\mathsf{XU}\, \chi$ there is a separate acceptance set, which contains all states which do not have an obligation for eventually satisfying this subformula or satisfy it in the sense that χ is an obligation too.

Theorem 4.4 (Gerth-Peled-Vardi-Wolper, [49])**.** Let P be a finite set of propositional variables and φ an LTL formula over P with n subformulas and k XU-subformulas. Then $\mathscr{A}[\varphi]$ is a generalized Büchi automaton with 2^n states and k acceptance sets such that $\mathscr{L}(\mathscr{A}[\varphi]) = \mathscr{L}(\varphi)$.

Let P be a finite set of propositional variables and φ an LTL formula over P in positive normal form. The generalized Büchi automaton for φ with respect to P, denoted $\mathscr{A}[\varphi]$, is defined by

$$\mathscr{A}[\varphi] = (2^P, 2^{\mathrm{sub}(\varphi)}, Q_I, \Delta, \mathscr{F})$$

where a triple (Ψ, a, Ψ') with $\Psi, \Psi' \subseteq \mathrm{sub}(\varphi)$ and $a \in 2^P$ belongs to Δ if the following conditions are satisfied:

(i) $\mathsf{ff} \notin \Psi$,

(ii) $p \in \Psi$ iff $p \in a$, for every $p \in P$,

(iii) if $\psi \vee \chi \in \Psi$, then $\psi \in \Psi$ or $\chi \in \Psi$,

(iv) if $\psi \wedge \chi \in \Psi$, then $\psi \in \Psi$ and $\chi \in \Psi$,

(v) if $\psi \,\mathsf{XU}\, \chi \in \Psi$, then $\chi \in \Psi'$ or $\{\psi, \psi \,\mathsf{XU}\, \chi\} \subseteq \Psi'$,

(vi) if $\psi \,\mathsf{XR}\, \chi$, then $\{\psi, \chi\} \subseteq \Psi'$ or $\{\chi, \psi \,\mathsf{XR}\, \chi\} \subseteq \Psi'$,

and where

$$Q_I = \{\Psi \subseteq \mathrm{sub}(\varphi) \colon \varphi \in \Psi\},$$
$$\mathscr{F} = \{F_{\psi \mathsf{XU} \chi} \colon \psi \,\mathsf{XU}\, \chi \in \mathrm{sub}(\varphi)\},$$

with $F_{\psi \mathsf{XU} \chi}$ defined by

$$F_{\psi \mathsf{XU} \chi} = \{\Psi \subseteq \mathrm{sub}(\varphi) \colon \chi \in \mathrm{sub}(\varphi) \text{ or } \psi \,\mathsf{XU}\, \chi \notin \Psi\}.$$

FIGURE 17. From LTL to generalized Büchi automata

Proof. We first show $\mathscr{L}(\mathscr{A}[\varphi]) \subseteq \mathscr{L}(\varphi)$. Let $u \in \mathscr{L}(\mathscr{A}[\varphi])$ and let r be an accepting run of $\mathscr{A}[\varphi]$ on u. We claim that for every i, if $\psi \in r(i)$, then $u, i \models \psi$. The proof is by induction on the structure of ψ. If $\psi = \mathsf{tt}$, $\psi = \mathsf{ff}$, $\psi = p$, or $\psi = \neg p$, then this follows directly from (i) or (ii). If $\psi = \chi \vee \zeta$, the claim follows from the induction hypothesis and (iii). Similarly, the claim holds for a conjunction.

Assume $\psi = \chi \,\mathsf{XR}\, \zeta$. Then (vi) tells us that

(a) $\chi \,\mathsf{XR}\, \zeta i \subseteq r(j)$ for every $j > i$ or

(b) there exists $j \geq i$ such that $\chi \,\mathsf{XR}\, \zeta \subseteq r(i')$ for i' with $i < i' < j$ and $\chi, \zeta \in r(j)$.

From the induction hypothesis and (a), we can conclude that we have $u, i' \models \zeta$ for all $i' > i$, which means $u, i \models \psi$. Similarly, from the induction hypothesis and (b), we can conclude that we have $u, i' \models \zeta$ for all i' such that $i < i' \leq j$ and $u, j \models \chi$, which implies $u, i \models \psi$, too.

Finally, assume $\psi = \chi \,\mathsf{XU}\, \zeta$. From (v), we obtain that

(a) $\chi \,\mathsf{XU}\, \zeta \in r(j)$ for all $j > i$ or

(b) there exists j such that $\chi \,\mathsf{XU}\, \zeta \in r(i')$ for all i' with $i < i' < j$ and $\zeta \in r(j)$.

Just as with XR, we obtain $u, i \models \psi$ from the induction hypothesis and (b). So we only need to show that if (a) occurs, we also have (b). Since r is accepting, there is some $\Psi \in F_{\chi \mathsf{XU} \zeta}$ such that $r(j) = \Psi$ for infinitely many j. Assuming (a), we can can conclude $\zeta \in \Psi$, which, by induction hypothesis, means we also have (b).

For the other inclusion, $\mathscr{L}(\varphi) \subseteq \mathscr{L}(\mathscr{A}[\varphi])$, we simply show that for a given u such that $u \models \varphi$ the word r defined by $r(i) = \{\psi \in \mathrm{sub}(\varphi) \colon u, i \models \psi\}$ is an accepting run of $\mathscr{A}[\varphi]$ on u. To this end, we need to show that

(a) r starts with an initial state,

(b) $(r(i), u(i), r(i+1)) \in \Delta$ for all i, and

(c) $r(i) \in F_{\psi \mathsf{XU} \chi}$ for infinitely many i, for every formula $\psi \,\mathsf{XU}\, \chi \in \mathrm{sub}(\varphi)$.

That (a) is true follows from the assumption $u \models \varphi$. Condition (b) is true simply because of the semantics of LTL. To see that (c) is true, let $\psi \,\mathsf{XU}\, \chi \in \mathrm{sub}(\varphi)$. We distinguish two cases. First, assume there exists i such that $u, j \not\models \chi$ for all $j > i$. Then $u, j \not\models \psi \,\mathsf{XU}\, \chi$ for all $j \geq i$, hence $r(j) \in F_{\psi \mathsf{XU} \chi}$ for all $j \geq i$, which is enough. Second, assume there are infinitely many i such that $u, i \models \chi$. Then $\chi \in r(i)$ for the same values of i, which is enough, too. Q.E.D.

Generalized Büchi automata can be converted into equivalent Büchi automata in a straightforward fashion. The idea is to check that every acceptance set is visited infinitely often by visiting these sets one after the other, in a fixed order, and repeating this process over and over again. In Figure 18, a respective construction is described. The second component of the state space is a counter which is used to keep track of the acceptance set to be visited next. When this counter reaches its maximum, every acceptance set has been visited once, and it can be reset.

Let $\mathscr{A}$ be a generalized Büchi automaton with $\mathscr{F} = \{F_0, \ldots, F_{k-1}\}$. The Büchi automaton $\mathscr{A}^{\mathrm{BA}}$ is defined by

$$\mathscr{A}^{\mathrm{BA}} = (A, Q \times \{0, \ldots, k\}, Q_I, \Delta', Q \times \{k\})$$

where Δ' contains for every $(q, a, q') \in \Delta$ the following transitions:

- $((q, k), a, (q', 0))$,
- $((q, i), a, (q', i))$ for every $i < k$,
- $((q, i), a, (q', i + 1))$ for every $i < k$ such that $q' \in F_i$.

FIGURE 18. From generalized Büchi to ordinary Büchi automata

Remark 4.5. Let $\mathscr{A}$ be a generalized Büchi automaton with n states and k acceptance sets. Then $\mathscr{A}^{\mathrm{BA}}$ is an equivalent Büchi automaton with at most $(k + 1)n$ states.

Corollary 4.6 (Vardi-Wolper, [124, 126])**.** There exists an effective procedure that given an LTL formula φ with n states and k XU-subformulas outputs a Büchi automaton $\mathscr{A}$ with at most $(k + 1)2^n$ states such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\varphi)$.

4.3 From LTL to alternating automata

The above translation from LTL into Büchi automata serves our purposes perfectly. We can use it to derive all the desired results about the complexity of the problems we are interested in, satisfiability, model checking, and realizability, as will be shown in the next subsection. There is, however, a translation using alternating automata, which is interesting in its own right. The motivation behind considering such a translation is to pass from the logical framework to the automata-theoretic framework in an as simple as possible fashion (to be able to apply powerful automata-theoretic tools as early as possible).

Alternating automata are provided with a feature to spawn several copies of themselves while running over a word. Formally, an alternating Büchi automaton is a tuple

$$\mathscr{A} = (P, Q, q_I, \delta, F)$$

where P, Q, and q_I are as usual, F is a Büchi acceptance condition, and δ is a function which assigns to each state q a transition condition, where every transition condition $\delta(q)$ is a positive boolean combination of formulas of the form p and $\neg p$, for $p \in P$, and $\bigcirc q$, for $q \in Q$. More precisely, the set of transition conditions over P and Q, denoted $\mathrm{TC}(P, Q)$, is the smallest set such that

(i) $\mathsf{tt}, \mathsf{ff} \in \mathrm{TC}(P, Q)$,

(ii) $p, \neg p \in \mathrm{TC}(P, Q)$ for every $p \in P$,

(iii) $\bigcirc q \in \mathrm{TC}(P, Q)$ for every $q \in Q$,

(iv) $\gamma \wedge \gamma', \gamma \vee \gamma' \in \mathrm{TC}(P, Q)$ for $\gamma, \gamma' \in \mathrm{TC}(P, Q)$.

A run of such an automaton on a word $u \in (2^P)^\omega$ is a tree $\mathscr{R}$ labeled with elements from $(Q \cup \mathrm{TC}(P, Q)) \times \omega$ such that $l^{\mathscr{R}}(\mathrm{root}(\mathscr{R})) = (q_I, 0)$ and the following conditions are satisfied for every $v \in V^{\mathscr{R}}$, assuming $l^{\mathscr{R}}(v) = (\gamma, i)$:

(i) $\gamma \neq \mathsf{ff}$,

(ii) if $\gamma = p$ for some $p \in P$, then $p \in u(i)$,

(iii) if $\gamma = \neg p$ for some $p \in P$, then $p \notin u(i)$.

(iv) if $\gamma = q$, then v has a successor v' such that $l^{\mathscr{R}}(v') = (\delta(q), i)$,

(v) if $\gamma = \bigcirc q'$, then v has a successor v' such that $l^{\mathscr{R}}(v') = (q', i + 1)$,

(vi) if $\gamma = \gamma_0 \wedge \gamma_1$, then v has successors v_0 and v_1 such that $l^{\mathscr{R}}(v_j) = (\gamma_j, i)$ for $j < 2$,

(vii) if $\gamma = \gamma_0 \vee \gamma_1$, then there exists $j < 2$ such that v has a successor v' with $l^{\mathscr{R}}(v') = (\gamma_j, i)$.

An infinite branch b of $\mathscr{R}$ is accepting if there are infinitely many i such that $l^{\mathscr{R}}(b(i)) \in F \times \omega$, in other words, there are infinitely many vertices with a final state in the first component of their labeling. The run is accepting if every infinite branch of it is accepting.

As a simple example, consider the language L_{10} over 2^P where $P = \{p\}$ which contains all words u satisfying the following condition: There exists some number i such that $p \in u(j + 10)$ for all $j \geq i$ with $p \in u(j)$. If we wanted to construct a nondeterministic automaton for this language, we could not do with less than 1000 states, but there is a small alternating automaton that recognizes this language. It simply guesses the right position i and for each position j it spawns off a copy of itself checking that after 10 further steps p holds true again. The details are given in Figure 19.

The automaton has states $q_I, q_0, q_1, \ldots, q_{10}$ where q_0 is the only final state and the transition function δ is defined by

- $\delta(q_I) = \bigcirc q_I \vee \bigcirc q_0$,
- $\delta(q_0) = \bigcirc q_0 \wedge ((p \wedge \bigcirc q_1) \vee \neg p)$,
- $\delta(q_i) = \bigcirc q_{i+1}$ for all i such that $0 < i < 10$,
- $\delta(q_{10}) = p$.

FIGURE 19. Example for an alternating automaton

It is (vi) from above which forces the tree to become a real tree, that is, it requires that a vertex has two successors (unless $\gamma_0 = \gamma_1$). So this is the condition that makes the automaton alternating: For a run to be accepting, both alternatives have to be pursued.

The translation from LTL to alternating Büchi automata, given in Figure 20, is straightforward as it simply models the semantics of LTL. It exploits the fact that $\psi \mathsf{XU} \chi$ and $\psi \mathsf{XR} \chi$ are equivalent to $\mathsf{X}\chi \vee (\mathsf{X}\psi \wedge \mathsf{X}(\psi \mathsf{XU} \chi))$ and $\mathsf{X}\chi \wedge (\mathsf{X}\psi \vee \mathsf{X}(\psi \mathsf{\ XR\ } \chi))$, respectively. Note that we use the notation $[\psi]$ to distinguish subformulas of φ from transition conditions ($p_0 \wedge p_1$ is different from $[p_0 \wedge p_1]$).

The transition function of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ has an interesting property, which we want to discuss in detail. Let $\leq$ be any linear ordering which extends the partial order on Q defined by $[\psi] \leq [\chi]$ if $\psi \in \mathrm{sub}(\chi)$. For every $\psi \in \mathrm{sub}(\varphi)$ and every $[\chi]$ occurring in $\delta([\psi])$, we have $[\chi] \leq [\psi]$. Following Gastin and Oddoux [48], we call an automaton satisfying this property a very weak alternating automaton.

The transition function of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ has an even stronger structural property, which we explain next. For a given symbol $a \in 2^P$, a transition condition γ, a state $q \in Q$, and a set $Q' \subseteq Q$, we define what it means that Q' is an a-successor of q with respect to γ, denoted $q \rightarrow^{a,\gamma} Q'$. This is defined inductively:

- $q \rightarrow^{a,\mathtt{tt}} \varnothing$,
- $q \rightarrow^{a,p} \varnothing$ if $p \in a$, and, similarly, $q \rightarrow^{a,\neg p} \varnothing$ if $p \notin a$,
- $q \rightarrow^{a,\bigcirc q'} \{q'\}$,
- $q \rightarrow^{a,\gamma_0 \vee \gamma_1} Q'$ if $q \rightarrow^{a,\gamma_0} Q'$ or $q \rightarrow^{a,\gamma_1} Q'$,

Let φ be an LTL formula in positive normal form over P and Q the set which contains for each $\psi \in \mathrm{sub}(\varphi)$ an element denoted $[\psi]$. The automaton $\mathscr{A}^{\mathrm{alt}}[A]$ is defined by

$$\mathscr{A}^{\mathrm{alt}}[\varphi] = (P, Q, [\varphi], \delta, F)$$

where

$$\begin{aligned} \delta([\mathsf{tt}]) &= \mathsf{tt}, & \delta([\mathsf{ff}]) &= \mathsf{ff}, \\ \delta([p]) &= p, & \delta([\neg p]) &= \neg p, \\ \delta([\psi \vee \chi]) &= \delta([\varphi]) \vee \delta([\psi]), & \delta([\psi \wedge \chi]) &= \delta([\varphi]) \wedge \delta([\psi]), \end{aligned}$$

$$\begin{aligned} \delta([\psi\, \mathsf{XU}\, \chi]) &= \bigcirc[\chi] \vee (\bigcirc[\psi] \wedge \bigcirc[\psi\, \mathsf{XU}\, \chi]), \\ \delta([\psi\, \mathsf{XR}\, \chi]) &= \bigcirc[\psi] \wedge (\bigcirc[\psi] \vee \bigcirc[\psi\, \mathsf{XR}\, \chi]), \end{aligned}$$

and F contains all the elements $[\psi] \in Q$ where ψ is not a XU-formula.

FIGURE 20. From an LTL formula to an alternating automaton

- $q \to^{a,\gamma_0 \wedge \gamma_1} Q'$ if there exists $Q_0, Q_1 \subseteq Q$ such that $Q' = Q_0 \cup Q_1$, $q \to^{a,\gamma_0} Q_0$, and $q \to^{a,\gamma_1} Q_1$.

Note that $q \to^{a,\gamma} Q'$ has a natural interpretation in terms of runs. If a vertex v of a run is labeled (q, i) and Q' is the set of all states q' such that $(q', i+1)$ is a label of a descendant of v, then $q \to^{a,\gamma} Q'$, provided, of course, that the run is minimal, which we can and will henceforth assume without loss of generality.

We use $q \to^a Q'$ as an abbreviation for $q \to^{a,\delta(q)} Q'$. We say a state q is persistent if there exists Q' such that $q \in Q'$ and $q \to^a Q'$ for some letter a.

Using the new notation, we can give an equivalent definition of being a very weak alternating automaton. It simply means that there exists a linear ordering $\leq$ on the states of the automaton such that if $q \to^a Q'$, then $q' \leq q$ for all $q' \in Q'$.

The automaton $\mathscr{A}^{\mathrm{alt}}[\varphi]$ has the following property. For every persistent state $q \in F$ there exists a state q' such that

(i) $q \to^a \{q'\}$ for every letter a and

(ii) whenever $q \to^a Q'$, then either $q \in Q'$ or $Q' = \{q'\}$.

(Every $q \notin F$ is of the form $[\psi \mathsf{XU} \chi]$, which means that we can choose $q' = [\chi]$.) We call very weak alternating automata that have this property ultra weak alternating automata and a state as q' above a discharging state for q and denote it by q^{d}.

Lemma 4.7. Let φ be an LTL formula with n subformulas. Then $\mathscr{A}^{\mathrm{alt}}[\varphi]$ is an ultra weak alternating automaton with n states such that $\mathscr{L}(\mathscr{A}^{\mathrm{alt}}[\varphi]) = \mathscr{L}(\varphi)$.

Proof. We only need to prove its correctness, which we do by an induction on the structure of φ. We start with a simple observation. Let $\mathscr{R}$ be an accepting run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u and $v \in V^{\mathscr{R}}$ labeled $([\psi], i)$ for some $\psi \in \mathrm{sub}(\varphi)$. Then $\mathscr{R}{\downarrow}v$ can be turned into an accepting run of $\mathscr{A}^{\mathrm{alt}}[\psi]$ on $u[i, *)$ by changing each second component j of a vertex label to $j - i$. Clearly, for this to be true $\mathscr{R}$ needs to be minimal (see above).

For the induction base, first assume $\varphi = \mathsf{tt}$ or $\varphi = \mathsf{ff}$. There is nothing to show. Second, assume $\varphi = p$ and suppose $u \models \varphi$. Then $p \in u(0)$, that is, the two-vertex tree where the root is labeled $([p], 0)$ and its only successor is labeled $(p, 0)$ is an accepting run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u. Conversely, if $\mathscr{R}$ is a (minimal) run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u, then $\mathscr{R}$ has two vertices labeled $([p], 0)$ and $(p, 0)$, respectively. This implies $p \in u(0)$, which, in turn, implies $u \models \varphi$. An analogous argument applies to $\neg p$.

In the inductive step, first assume $\varphi = \psi_0 \wedge \psi_1$. If there exists an accepting run $\mathscr{R}$ of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u, then, because of $\delta([\varphi]) = \delta([\psi_0]) \wedge \delta([\psi_1])$, the root has successors v_0 and v_1 such that $l^{\mathscr{R}}(v_i) = (\delta([\psi_i]), 0)$. For every i, we can turn $\mathscr{R}{\downarrow}v_i$ into an accepting run $\mathscr{R}_i$ of $\mathscr{A}^{\mathrm{alt}}[\psi_i]$ on u by adding a new root labeled $([\psi_i], 0)$. By induction hypothesis, we obtain $u \models \psi_i$ for every i, hence $u \models \varphi$. Conversely, assume $u \models \varphi$. Then $u \models \psi_i$ for $i < 2$, and, by induction hypothesis, there exist accepting runs $\mathscr{R}_i$ of $\mathscr{A}^{\mathrm{alt}}[\psi_i]$ on u for $i < 2$. These runs can be turned into an accepting run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u by simply making their vertex sets disjoint, removing their roots, and adding a new common root labeled $([\varphi], 0)$.

A similar argument applies to formulas of the form $\psi_0 \vee \psi_1$.

Next, assume $\varphi = \psi \mathsf{XU} \chi$. Suppose $\mathscr{R}$ is an accepting run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u and let v_0 be the root of this run. Also, let $u_i = u[i, *)$ for every i. Then, by definition of accepting run, $l^{\mathscr{R}}(v_0) = ([\psi \mathsf{XU} \chi], 0)$. From the definition of the transition function we can conclude that v_0 has a successor, say v_1, which is labeled by $(\bigcirc[\chi] \vee (\bigcirc[\psi] \wedge \bigcirc[\psi \mathsf{XU} \chi]), 0)$, which, in turn, has a successor, say v_2, which is labeled by either $(\bigcirc[\chi], 0)$ or $(\bigcirc[\psi] \wedge \bigcirc[\psi \mathsf{XU} \chi], 0)$. In the first case, there is a further successor labeled $([\chi], 1)$ and we obtain $u_1 \models \chi$ from the induction hypothesis, hence, $u \models \varphi$. In the second case, we know there exist successors v_3 and v_3' of v_2 labeled $(\bigcirc[\psi \mathsf{XU} \chi], 0)$ and $(\bigcirc[\psi], 0)$, respectively, which themselves have successors v_4 and v_4' labeled

$([\psi \,\mathsf{XU}\, \chi], 1)$ and $([\psi], 1)$, respectively. By induction hypothesis, we obtain $u_1 \models \psi$. Applying the same arguments as before, we find that either there is a vertex labeled $([\chi], 2)$ or there are vertices v_8 and v_8' labeled $[(\psi \,\mathsf{XU}\, \chi, 2)]$ and $([\psi], 2)$, respectively. In the first case, we get $u \models \varphi$ because we also know $u_1 \models \psi$, whereas in the second case we can again apply the same arguments as before. Continuing in this fashion, we find that the only case which remains is the one where we have an infinite sequence of vertices $v_4, v_8, v_{12}, \ldots$ on the same branch and every vertex with label in $Q \times \omega$ is labeled $([\varphi], i)$, which means that this branch is not accepting—a contradiction.

For the other direction, assume $u \models \varphi$ and use the same notation as before. Then there is some $j > 0$ such that $u_j \models \chi$ and $u_i \models \psi$ for all i with $0 < i < j$. By induction hypothesis, there are accepting runs $\mathscr{R}_i$ for i with $0 < i < j$ of $\mathscr{A}^{\mathrm{alt}}[\psi]$ on u_i and an accepting run $\mathscr{R}_j$ of $\mathscr{A}^{\mathrm{alt}}[\chi]$ on u_j. Assume that $v_1, \ldots, v_j$ are the roots of these trees and assume that their sets of vertices are pairwise disjoint. Then we can construct an accepting run $\mathscr{R}$ for $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u as follows. The vertices of $\mathscr{R}$ are the vertices of the $\mathscr{R}_k$'s and, in addition, the vertices $w_0, w_0', w_0'', w_0''', \hat{w}_0, w_1, \ldots, w_{j-1}, w_{j-1}', w_{j-1}''$. The labeling is as follows:

- w_i is labeled $([\varphi], i)$ for $i < j$,
- w_i' is labeled $(\bigcirc[\chi] \vee (\bigcirc[\psi] \wedge \bigcirc[\varphi]), i)$ for $i < j$,
- w_i'' is labeled $(\bigcirc[\psi] \wedge \bigcirc[\varphi], i)$ for $i < j - 1$,
- w_i''' is labeled $(\bigcirc[\varphi], i)$ for $i < j - 1$,
- $\hat{w}_i$ is labeled $(\bigcirc[\psi], i)$ for $i < j - 1$, and
- w_j'' is labeled $(\bigcirc[\chi], j - 1)$.

The tree $\mathscr{R}$ has all edges from the $\mathscr{R}_k$'s and, in addition,

- edges such that $w_0 w_0' w_0'' w_0''' \ldots w_{j-1} w_{j-1}' w_{j-1}'' v_j$ is a path and
- edges $(w_i', \hat{w}_i)$ and $(\hat{w}_i, v_i)$ for every $i < j$.

This yields an accepting run of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ on u.

Finally, XR can be dealt with in a similar fashion. Q.E.D.

It is not very difficult to translate alternating Büchi automata into nondeterministic Büchi automata, as was shown by Miyano and Hayashi [87], but it yields a worse upper bound compared to a translation from ultra weak alternating automata to Büchi automata. This is why we present the

Let $\mathscr{A}$ be an ultra weak alternating automaton over a finite set P of propositional variables. The generalized Büchi automaton for $\mathscr{A}$, denoted $\mathscr{A}^{\mathrm{gBA}}$, is defined by

$$\mathscr{A}^{\mathrm{gBA}} = (2^P, 2^Q, \{q_I\}, \Delta, \mathscr{F})$$

where

- the transition relation Δ contains a transition (Q', a, Q'') if for every $q \in Q'$ there exists a set Q_q such that $q \to^{a,\delta(q)} Q_q$ and $\bigcup_{q \in Q'} Q_q \subseteq Q''$ and
- the set $\mathscr{F}$ of acceptance sets contains for every $q \notin F$ the set F_q defined by $\{Q' \subseteq Q \colon q^{\mathrm{d}} \in Q' \text{ or } q \notin Q'\}$.

FIGURE 21. From ultra weak to generalized Büchi automata

latter. Another advantage of this translation is that it can be simplified by going through alternating generalized Büchi automata.

The main idea of the translation from ultra weak alternating automata to (generalized) Büchi automata is to use a powerset construction to keep track of the individual branches of an accepting run of the alternating automaton. There are two technical problems that we face in the translation. First, we need to take care of the vertices in the runs which are not labeled with a state (but with a transition condition), and, second, we need to take care of the acceptance condition. The first problem is similar to removing ε-transitions and the second problem can be solved by using the fact that the automata are ultra weak. The entire construction is described in Figure 21.

Lemma 4.8. Let $\mathscr{A}$ be an ultra weak alternating automaton with n states and k final states. Then $\mathscr{A}^{\mathrm{gBA}}$ is an equivalent generalized Büchi automaton with 2^n states and k acceptance sets.

Proof. The claim about the number of states and the number of acceptance sets is obvious. We only need to show that the translation is correct.

First, assume $u \in \mathscr{L}(\mathscr{A})$. Then there is an accepting run $\mathscr{R}$ of $\mathscr{A}$ on u (which we assume to be minimal again). We say a vertex $v \in V^{\mathscr{R}}$ is a state vertex if the first component of its label is a state. Let $\mathscr{R}'$ be the tree which is obtained from $\mathscr{R}$ by "removing" the non-state vertices while keeping their edges. Formally, $\mathscr{R}'$ is constructed inductively as follows. We start with the root of $\mathscr{R}$, which is a state vertex by definition. Then, once

we have a vertex v of $\mathscr{R}'$, we add all state vertices v' of $\mathscr{R}$ as successors of v to $\mathscr{R}'$ which can be reached from v in $\mathscr{R}$ via a path without state vertices (not counting the first and last vertex).

The tree $\mathscr{R}'$ has the following property. When v is a vertex labeled (q,i) and $\{v_0,\ldots,v_{m-1}\}$ is the set of its successors where v_j is labeled (q_j,i_j), then $q \to^{u(i)} \{q_0,\ldots,q_{m-1}\}$ and $i_j = i+1$ for every $j < m$. This is because the definition of $\to^{a,\gamma}$ simply models the requirements of a run.

Using the above property of $\mathscr{R}'$ we can easily construct a run r of $\mathscr{A}^{\mathrm{gBA}}$ on u as follows. We simply let $r(i)$ be the set of all q such that there exists a vertex v in $\mathscr{R}'$ labeled (q,i). By definition of $\mathscr{A}^{\mathrm{gBA}}$, this is a run. What remains to be shown is that r is an accepting run.

Assume $q \notin F$ and i is an arbitrary number. We have to show that there exists $j \geq i$ such that $r(j) \in F_q$. If there is some $j \geq i$ such that $q \notin r(j)$, this is true. So assume that $q \in r(j)$ for all $j \geq i$. By construction of $\mathscr{R}'$ there exists a vertex v_0 in $\mathscr{R}'$ which is labeled (q,i). If one of the successors of v_0 is labeled q^{d} in the first component, then $r(i+1) \in F_q$, which is enough. If, on the other hand, all successors are labeled distinct from q^{d} in their first component, then, since $\mathscr{A}$ is assumed to be ultra weak, one of the successors, say v_1, is labeled q in the first component. We can apply the same argument as before to v_1 now. We find that $r(i+2) \in F_q$ or we find a successor v_2 of v_1 with q in the first component of its label, too. If we continue like this and we do not find $r(j)$ such that $r(j) \in F_q$, we obtain an infinite path $v_0v_1\ldots$ in $\mathscr{R}'$ where every v_i is labeled q in the first component. This path can be prefixed such that it becomes a branch of $\mathscr{R}$, and this branch is not accepting—a contradiction to the assumption that $\mathscr{R}$ is accepting.

For the other direction, assume $u \in (2^P)^\omega$ is accepted by $\mathscr{A}^{\mathrm{gBA}}$ and let r be an accepting run of $\mathscr{A}^{\mathrm{gBA}}$ on u. For every i and every $q \in r(i)$, let Q^i_q be a set such that $q \to^{u(i),\delta(q)} Q^i_q$ for all $q \in r(i)$ and $\bigcup\{Q^i_q \colon q \in r(i)\} \subseteq r(i)$. By definition of $\mathscr{A}^{\mathrm{gBA}}$, such sets exist. For some combinations of q and i there might be several choices for Q^i_q. By convention, if $q^{\mathrm{d}} \in r(i+1)$, we let $Q^i_q = \{q^{\mathrm{d}}\}$, which is a possible choice since $\mathscr{A}$ is assumed to be ultra weak. Using these sets, we construct a tree $\mathscr{R}'$ from r inductively as follows. We start with the root and label it $(q_I,0)$. If we have a vertex v labeled (q,i), we add a successor to v for every $q' \in Q^i_q$ and label it $(q',i+1)$. By expanding $\mathscr{R}'$ according to the semantics of the transition conditions, we obtain a tree $\mathscr{R}$ which is a run of $\mathscr{A}$ on u. It remains to be shown that this run is accepting. Assume this is not the case. Then, because $\mathscr{A}$ is ultra weak, there is a non-final state q, a branch $v_0v_1\ldots$ of $\mathscr{R}'$, and a number i such that the label of v_j is (q,j) for all $j \geq i$. This implies $q \in Q^i_q$ for all $j \geq i$. Since r is accepting, we know that there exists $j > i$ such that $q \notin r(j)$ or $q^{\mathrm{d}} \in r(j)$. The first condition is an immediate contradiction. So

assume $q^{\mathrm{d}} \in r(j)$ for some $j > i$. Since we have $q \in r(j-1)$, we also have $Q_q^j = \{q^{\mathrm{d}}\}$ by construction—a contradiction. Q.E.D.

Combining the previous lemma and Remark 4.5 yields an alternative proof of Corollary 4.6. Very weak alternating automata are interesting for another reason, too:

Theorem 4.9 (Rohde, [103])**.** For every very weak alternating automaton $\mathscr{A}$ there exists an LTL formula φ such that $\mathscr{L}(\varphi) = \mathscr{L}(\mathscr{A})$.

This was also proved by Löding and Thomas [81] and a proof of it can be found in [30].

4.4 LTL satisfiability, model checking, and realizability

We can now return to the problems we are interested in, satisfiability, validity, model checking, and realizability.

Theorem 4.10 (Clarke-Emerson-Sistla, [27])**.** LTL satisfiability is PSPACE-complete.

Proof. Given an LTL formula φ over a set P of propositional variables, we construct a Büchi automaton equivalent to φ and check this automaton for nonemptiness. Clearly, this procedure is correct. To determine its complexity, we use the following simple fact from complexity theory.

(†) Let $f\colon A^* \to B^*$ be a function computable in PSPACE and $L \subseteq B^*$ a problem solvable in nondeterministic logarithmic space. Then $f^{-1}(P) \in$ PSPACE.

When we apply (†) to the situation where f computes the above Büchi automaton equivalent to φ and L is the problem whether a Büchi automaton accepts some word, then we obtain that our problem is in PSPACE.

For the lower bound, we refer the reader to [27] or [106]. Q.E.D.

For model checking, the situation is essentially the same as with S1S. When we are given a finite-state automaton $\mathscr{D}$ over the alphabet 2^P for some finite set P of propositional variables and φ is an LTL formula over P, we write $\mathscr{D} \models \varphi$ if $u \models \varphi$ for all $u \in \mathscr{L}(\mathscr{D})$. LTL model checking is the problem, given $\mathscr{D}$ and φ, to determine whether $\mathscr{D} \models \varphi$, that is, whether $\mathscr{L}(\mathscr{D}) \subseteq \mathscr{L}(\varphi)$.

Theorem 4.11. (Sistla-Clarke-Lichtenstein-Pnueli, [109, 80])

(1) LTL model checking is PSPACE-complete.

(2) Given a formula φ with n subformulas and a finite-state automaton $\mathscr{D}$ of size m, whether $\mathscr{D} \models \varphi$ holds can be checked in time $2^{O(n)}m$.

Proof. The same approach as in Section 2.1 yields the desired upper bounds. Given a finite set of propositional variables P, a finite-state automaton $\mathscr{D}$ over 2^P, and an LTL formula over P, we first construct the product $\mathscr{A} \times \mathscr{D}$ where $\mathscr{A}$ is a Büchi automaton equivalent to $\neg\varphi$. We have $\mathscr{L}(\mathscr{A} \times \mathscr{D}) = \varnothing$ if and only if $\mathscr{D} \models \varphi$. So, to conclude, we apply an emptiness test.

The number of states of the product is at most $(k+1)2^n \cdot m$ where n is the size of φ, the number k is the number of XU-formulas in φ (after transformation to positive normal form), and m is the number of states of $\mathscr{D}$. Using the same complexity-theoretic argument as in the proof of Theorem 4.10, we obtain part 1.

Part 2 follows from the fact that an emptiness test for a Büchi automaton can be carried out in time linear in the size of the automaton.

For the lower bound, we refer the reader to [27]. Q.E.D.

Finally, we turn to realizability, which is defined as with S1S (see Section 2.4). An LTL realizability instance is an LTL formula over a set $P = \{p_0, \ldots, p_{m-1}, q_0, \ldots, q_{n-1}\}$ of propositional variables. Just as earlier in this section, we interpret such formulas in words over $[2]_{m+n}$, which means that a solution of such an instance is a function $f\colon [2]_m^+ \to [2]_n$ satisfying the requirement known from the S1S setting, that is, $u^\frown v \models \varphi$ holds for every $u \in [2]_m^\omega$ and $v \in [2]_n^\omega$ defined by $v(i) = f(u[0,i])$ for every i.

We can use the same technique as in Section 3 to obtain the following result:

Theorem 4.12 (Pnueli-Rosner, [99])**.** LTL realizability is complete for doubly exponential time. Moreover, for every positive instance a finite-state machine realizing a finite-state solution can be computed within the same time bound.

Proof. Consider the following algorithm for solving a given instance φ over $\{p_0, \ldots, p_{m-1}, q_0, \ldots, q_{n-1}\}$. First, consider the game $\mathscr{G}[\varphi]$ which is obtained using the construction from Figure 10 with the S1S formula replaced by the LTL formula. Second, compute a Büchi automaton $\mathscr{A}$ equivalent to φ according to Corollary 4.6. Third, turn $\mathscr{A}$ into a deterministic parity automaton $\mathscr{B}$ according to 2.14. Fourth, let $\mathscr{G} = \mathscr{G}[\varphi] \times \mathscr{B}$ be the game obtained from expanding $\mathscr{G}[\varphi]$ by $\mathscr{B}$. Fifth, solve the game $\mathscr{G}$ using Theorem 2.21. Player 0 wins $\mathscr{G}$ if and only if φ is a positive instance of realizability.

To prove the desired complexity bound let n be the number of subformulas of φ and observe the following. The size of $\mathscr{A}$ is at most $(n+1)2^n$. Therefore, the worst-case size of $\mathscr{B}$ is $2^{O(2^n n \log n)}$ and $\mathscr{B}$ has at most $3(n+1)2^n$ priorities. Theorem 2.21 now gives the desired upper bound.

The additional claim about the finite-state solution follows from Lemmas 2.16 and 2.17. For the lower bound, see [104]. Q.E.D.

In the remainder of this section, we present an alternative approach to solving the realizability problem, which is interesting in its own right.

Let φ be an instance of the realizability problem as above. Formally, a solution of φ is a function $f\colon [2]_m^+ \to [2]_n$. Such a function is the same as a $[2]_m$-branching $[2]_n$-labeled tree (where the root label is ignored). In other words, the set of all solutions of a given instance of the realizability problem is a tree language. This observation transforms the realizability problem into the framework of tree languages and tree automata, and we can apply tree-automata techniques to solve it.

Let $t\colon [2]_m^* \to [2]_n$ be any $[2]_m$-branching $[2]_n$-labeled tree. The tree can be turned into a potential solution to the instance φ if the label of the root is forgotten. The resulting function is denoted by $t_{-\varepsilon}$. We set $\mathscr{L}_{\mathrm{sol}}(\varphi) = \{t\colon [2]_m^* \to [2]_n : t_{-\varepsilon} \text{ solves } \varphi\}$.

We next show that $\mathscr{L}_{\mathrm{sol}}(\varphi)$ is a tree language which can be recognized by a universal tree automaton. We need, however, a more general notion of universal tree automaton as in Section 3.3. Also, we need to massage the formula φ a little to arrive at a simple automata-theoretic construction.

A universal co-Büchi tree automaton with set of directions D is a tuple

$$(A, D, Q, q_I, \Delta, F)$$

where A, Q, q_I, and F are as usual, and where D is a finite set of directions and $\Delta \subseteq Q \times A \times D \times Q$ is a transition relation. Following the definition from Section 3.3, a word $r \in Q^\omega$ is said to be a run for branch $u \in D^\omega$ if $(r(i), t(u[0,i)), u(i), r(i+1)) \in \Delta$ for every i and $r(0) = q_I$. A tree is accepted if every $r \in Q^\omega$ which is a run for some branch satisfies the co-Büchi acceptance condition. The latter means that $r(i) \in F$ only for finitely many i.

The technical problem one faces when constructing an automaton for $\mathscr{L}_{\mathrm{sol}}(\varphi)$ is that a tree automaton has transitions of the form (q, a, d, q'), so, when applied to the above setting, in one transition the automaton consumes an output of the device we are looking for and the next input. For our construction it would be much better to have automata that in one transition consume an input and a corresponding output. Rather than modifying our standard automaton model, we resolve the issue on the logical side. For a given formula $\varphi = \varphi(p_0, \ldots, p_{m-1}, q_0, \ldots, q_{n-1})$ we consider the formula φ^{X} defined by

$$\varphi^{\mathsf{X}} = \varphi(p_0, \ldots, p_{m-1}, \mathsf{X}q_0, \ldots, \mathsf{X}q_{n-1}).$$

(Recall that X stands for the temporal operator "next".) This formula moves the output one position to the right, more precisely,

$$\mathscr{L}(\varphi) = \{d^0{}^\frown a^1 d^1{}^\frown a^2 \ldots : d^0 \frown a^0 d^1{}^\frown a^1 \cdots \in \mathscr{L}(\varphi^{\mathsf{X}})\}. \tag{1.2}$$

Let $\varphi = \varphi(p_0, \ldots, p_{m-1}, q_0, \ldots, q_{n-1})$ be an instance of the LTL realizability problem and $\mathscr{A}$ a Büchi automaton such that $\mathscr{L}(\mathscr{A}) = \mathscr{L}(\neg\varphi^{\mathsf{X}})$. The universal co-Büchi tree automaton for φ, denoted $\mathscr{A}^{\mathrm{real}}[\varphi]$, is defined by

$$\mathscr{A}^{\mathrm{real}}[\varphi] = ([2]_n, [2]_m, Q, q_I, \Delta', F)$$

where

$$\Delta' = \{(q, a, d, q') \colon (q, d^\frown a, q') \in \Delta\}.$$

FIGURE 22. From an LTL realizability instance to a universal tree automaton

A universal co-Büchi tree automaton for a given LTL formula φ as above is now easily constructed, as can be seen in Figure 22.

Lemma 4.13. Let $\varphi = \varphi(p_0, \ldots, p_{m-1}, q_0, \ldots, q_{n-1})$ be an instance of the LTL realizability problem. Then $\mathscr{L}(\mathscr{A}^{\mathrm{real}}[\varphi]) = \mathscr{L}_{\mathrm{sol}}(\varphi)$. Q.E.D.

Universal co-Büchi tree automata for D-branching trees as defined above are a special case of universal parity tree automata for D-branching trees, which can be turned into nondeterministic parity tree automata for D-branching trees in the same fashion as this was explained for automata on binary trees in Figure 16. The same is true for the emptiness test for parity tree automata on D-branching trees, which can be solved by constructing a parity game along the lines of the construction depicted in Figure 13 and solving this game.

4.5 Notes

The automata-theoretic decision procedure for LTL model checking described in this section has had a great practical impact, because it has been implemented in an industrial setting, see, for instance, [57], and used to verify real-world computing systems (mostly hardware). Much research has gone into improving the algorithm in several respects, but also into extending its applicability, for instance, more expressive logics and larger classes of devices have been looked at, see, for instance, [14, 25, 44, 70]. It is also noteworthy that LTL is the basis for industrial specification languages such as ForSpec [2] and PSL [34] and that the automata-theoretic approach underlies industrial implementations of specification languages [3].

An important aspect of this section is the use of alternating automata, which were introduced into the theory of automata on infinite objects by Muller and Schupp [90]. The only gain from this presented in the current section is Theorem 4.9, but this is probably the least important aspect in this context. What is more important is that weak alternating automata are as powerful as nondeterministic Büchi automata, which was proved by Kupferman and Vardi [72, 73]. This result motivated new research, which, for instance, brought about new complementation constructions [72, 73, 121]. As we see in the remaining two sections, alternation is even more important in the context of tree languages.

We refer to [123] for a collection of open algorithmic issues with regard to automata-theoretic LTL model checking.

5 Computation tree logic

Certain temporal properties of a system cannot be specified when runs of the system are considered separately, as we do this with LTL. For instance, when one wants to specify that no matter which state a system is in there is some way to get back to a default state, then this cannot be stated in LTL. The reason is that the property says something about how a run can evolve into different runs.

This observation motivates the introduction of specification logics that compensate for the lack of expressive power in this regard. The first logic of this type, called UB, was introduced by Ben-Ari, Manna, and Pnueli [7] in 1981. Another logic of this type is computation tree logic (CTL), designed by Clarke and Emerson [36], which is interpreted in the "computation tree" of a given transition system. This is the logic we study in this section, in particular, we study satisfiability and model checking for this logic.

Many of the proofs in this section are very similar to proofs in the previous section. In these cases, we only give sketches, but describe the differences in detail.

5.1 CTL and monadic second-order logic

CTL mixes path quantifiers and temporal operators in a way such that a logic arises for which model checking can be carried out in polynomial time. The syntax of CTL is as follows:

- tt and ff are CTL formulas,
- every propositional variable is a CTL formula,
- if φ is a CTL formula, then so is $\neg\varphi$,
- if φ and ψ are formulas, then so are $\varphi \vee \psi$, $\mathsf{E}(\varphi\,\mathsf{XU}\,\psi)$, and $\mathsf{A}(\varphi\,\mathsf{XU}\,\psi)$.

CTL formulas are interpreted in transition systems, which we introduce next. Such a system is a simple, state-based abstraction of a computing device. Formally, it is a tuple

$$\mathscr{S} = (P, S, \to, l)$$

where P is a finite set of propositional variables, S is a set of states, $\to \subseteq S \times S$ is a transition relation in infix notation, and $l\colon S \to 2^P$ is a labeling function assigning to each state which propositional variables are true in it. A computation of such a transition system starting in a state s is a word $u \in S^+ \cup S^\infty$ such that

(i) $u(0) = s$,

(ii) $u(i) \to u(i+1)$ for all i with $i + 1 < |u|$, and

(iii) u is maximal in the sense that if u is finite, then $u(*)$ must not have any successor.

Given a CTL formula φ, a transition system $\mathscr{S}$ over the same set P of propositional variables, and a state s of $\mathscr{S}$, it is defined whether φ holds true in $\mathscr{S}$ at s, denoted $\mathscr{S}, s \models \varphi$:

- $\mathscr{S}, s \models \mathsf{tt}$ and $\mathscr{S}, s \not\models \mathsf{ff}$,
- $\mathscr{S}, s \models p$ if $p \in l(s)$,
- $\mathscr{S}, s \models \neg\varphi$ if $\mathscr{S}, s \not\models \varphi$,
- $\mathscr{S}, s \models \psi \vee \chi$ if $\mathscr{S}, s \models \psi$ or $\mathscr{S}, s \models \chi$, for ψ and χ CTL formulas,
- $\mathscr{S}, s \models \mathsf{E}(\psi\, \mathsf{XU}\, \chi)$ if there exists a computation u of $\mathscr{S}$ starting at s and $j > 0$ such that $\mathscr{S}, u(j) \models \chi$ and $\mathscr{S}, u(i) \models \psi$ for all i with $0 < i < j$.
- $\mathscr{S}, s \models \mathsf{A}(\psi\, \mathsf{XU}\, \chi)$ if for all computations u of $\mathscr{S}$ starting at s there exists $j > 0$ such that $\mathscr{S}, u(j) \models \chi$ and $\mathscr{S}, u(i) \models \psi$ for all i with $0 < i < j$.

Just as with LTL, other operators can be defined:

- "in all computations always" is defined by $\mathsf{AG}\varphi = \varphi \wedge \neg\mathsf{E}(\mathsf{tt}\, \mathsf{XU}\, \neg\varphi)$,
- "in some computation eventually" is defined by $\mathsf{EF}\varphi = \varphi \vee \mathsf{E}(\mathsf{tt}\, \mathsf{XU}\, \varphi)$.

An interesting property one can express in CTL is the one discussed above, namely that from every state reachable from a given state a distinguished state, indicated by the propositional variable p_d, can be reached:

$$\mathsf{AG}\,\mathsf{EF}p_d. \tag{1.3}$$

Another property that can be expressed is that every request, indicated by the propositional variable p_r, is eventually acknowledged, indicated by the propositional variable p_a:

$$\mathsf{AG}(p_r \to \mathsf{AX}\,\mathsf{AF}p_a). \tag{1.4}$$

It is interesting to compare the expressive power of CTL with that of LTL. To this end, it is reasonable to restrict the considerations to infinite computations only and to say that a CTL formula φ and an LTL formula ψ are equivalent if for every transition system $\mathscr{S}$ and every state $s \in S$ the following holds: $\mathscr{S}, s \models \varphi$ iff $l(u(0))l(u(1))\ldots \models \psi$ for all infinite computations u of $\mathscr{S}$ starting in s.

The second property from above can be expressed easily in LTL, namely by the formula $\mathsf{G}(p_r \to \mathsf{XF}p_a)$, that is, this formula and (1.4) are equivalent. Clarke and Draghicescu showed that a CTL property is equivalent to some LTL formula if and only if it is equivalent to the LTL formula obtained by removing the path quantifiers [26]. But it is not true that every LTL formula which can be expressed in CTL is expressible by a CTL formula which uses universal path quantifiers only. This was shown by Bojanczyk [10].

An LTL formula which is not expressible in CTL is

$$\mathsf{GF}p, \tag{1.5}$$

which was already pointed out by Lamport [77].

In order to be able to recast satisfiability and model checking in a (tree) automata setting, it is crucial to observe that CTL formulas cannot distinguish between a transition system and the transition system obtained by "unraveling" it. Formally, the unraveling of the transition system $\mathscr{S}$ at state $s \in S$, denoted $\mathscr{T}_s(\mathscr{S})$, is the tree inductively defined by:

- s is the root of $\mathscr{T}_s(\mathscr{S})$,
- if $v \in S^+$ is an element of $V^{\mathscr{T}_s(\mathscr{S})}$ and $v(*) \to s'$, then $vs' \in V^{\mathscr{T}_s(\mathscr{S})}$ and $(v, vs') \in E^{\mathscr{T}_s(\mathscr{S})}$,
- $l^{\mathscr{T}_s(\mathscr{S})}(v) = l^{\mathscr{S}}(v(*))$ for every $v \in V^{\mathscr{T}_s(\mathscr{S})}$.

Henceforth, a tree with labels from 2^P, such as the unraveling of a transition system, is viewed as a transition system in the canonical way. When we

interpret a CTL formula in a tree and do not indicate a vertex, then the formula is interpreted at the root of the tree.

The formal statement of the above observation can now be phrased as follows.

Lemma 5.1. For every CTL formula φ, transition system $\mathscr{S}$, and state $s \in S$,

$$\mathscr{S}, s \models \varphi \qquad \text{iff} \qquad \mathscr{T}_s(\mathscr{S}) \models \varphi.$$

Proof. This can be proved by a straightforward induction on the structure of φ, using a slightly more general claim:

$$\mathscr{S}, s' \models \varphi \qquad \text{iff} \qquad \mathscr{T}_s(\mathscr{S}), v \models \varphi$$

for every state $s' \in S$ and every vertex v of $\mathscr{T}_s(\mathscr{S})$ where $v(*) = s'$. Q.E.D.

The previous lemma says that we can restrict attention to trees, in particular, a CTL formula is satisfiable if and only if there is a tree which is a model of it. So when we translate CTL formulas into logics on trees which satisfiability is decidable for, then we also know that CTL satisfiability is decidable.

We present a simple translation of CTL into monadic second-order logic. There is, however, an issue to be dealt with: S2S formulas specify properties of binary trees, but CTL is interpreted in transition systems where each state can have more than just two successors. A simple solution is to use a variant of S2S which allows any number of successors but has only a single successor predicate, suc. Let us denote the resulting logic by SUS. As with LTL, we identify the elements of 2^P for $P = \{p_0, \ldots, p_{n-1}\}$ with the elements of $[2]_n$.

Proposition 5.2. Let $P = \{p_0, \ldots, p_{n-1}\}$ be an arbitrary finite set of propositional variables. For every CTL formula φ over P an SUS formula $\tilde{\varphi} = \tilde{\varphi}(X_0, \ldots, X_{n-1})$ can be constructed such that $\mathscr{T} \models \varphi$ if and only if $\mathscr{T} \models \tilde{\varphi}$ for all trees $\mathscr{T}$ over 2^P (or $[2]_n$).

Proof. What we actually prove is somewhat stronger, analogous to the proof for LTL. We construct a formula $\hat{\varphi} = \hat{\varphi}(X_0, \ldots, X_{n-1}, x)$ such that $\mathscr{T}, v \models \varphi$ if and only if $\mathscr{T}, v \models \hat{\varphi}$ for all trees $\mathscr{T}$ and $v \in V^{\mathscr{T}}$. We can then set $\tilde{\varphi} = \exists x(\varphi_{root}(x) \wedge \hat{\varphi})$ where $\varphi_{root}(x) = \forall y(\neg \mathrm{suc}(y, x))$ specifies that x is the root.

For the induction base, assume $\varphi = p_i$. We can set $\hat{\varphi}$ to $x \in X_i$. Similarly, for $\varphi = \neg p_i$ we can set $\hat{\varphi}$ to $\neg x \in X_i$.

In the inductive step, we consider only one of the interesting cases, namely where $\varphi = \mathsf{A}(\psi \,\mathsf{XU}\, \chi)$. We start with a formula $\varphi_{closed} = \varphi_{closed}(X)$

which is true if every element of X has a successor in X provided it has a successor at all:

$$\varphi_{closed} = \forall x(x \in X \wedge \exists y(\mathrm{suc}(x,y)) \rightarrow \exists y(\mathrm{suc}(x,y) \wedge y \in X)).$$

We next write a formula $\varphi_{path}(x,X)$ which is true if X is a maximum path starting in x:

$$\varphi_{path} = x \in X \wedge \varphi_{closed}(X) \wedge \forall Y(x \in Y \wedge \varphi_{closed}(Y) \wedge Y \subseteq X \rightarrow X = Y).$$

We can then set

$$\hat{\varphi} = \forall X(\varphi_{path}(x,X) \rightarrow \exists z(z \in X \wedge \neg z = x \wedge \hat{\chi}(z) \wedge \forall y(x < y < z \rightarrow \hat{\psi}(y))).$$

The other CTL operators can be dealt with in a similar fashion. Q.E.D.

The desired decidability result now follows from the following result on SUS.

Theorem 5.3 (Walukiewicz, [128])**.** SUS satisfiability is decidable.

This result can be proved just as we proved the decidability of satisfiability for S2S, that is, using an analogue of Rabin's Theorem. This analogue will use a different kind of tree automaton model which takes into account that the branching degree of the trees considered is unbounded and that there is one predicate for all successors. More precisely, a transition in such an automaton is of the form $(q, a, Q^{\mathsf{E}}, Q^{\mathsf{A}})$ where $Q^{\mathsf{E}}, Q^{\mathsf{A}} \subseteq Q$. Such a transition is to be read as follows: If the automaton is in state q at a vertex labeled a, then for every $q' \in Q^{\mathsf{E}}$ there exists exactly one successor that gets assigned q' and all the successors that do not get assigned any state in this fashion get assigned exactly one state from Q^{A}. In particular, if $Q^{\mathsf{E}} = Q^{\mathsf{A}} = \varnothing$, then the vertex must not have a successor. In [128], Walukiewicz actually presents a theorem like Büchi's and Rabin's: He shows that there is a translation in both directions, from SUS formulas to such automata and back.

Corollary 5.4. CTL satisfiability and model checking are decidable.

That model checking is decidable follows from the simple observation that in SUS one can define the unraveling of every finite transition system.

We conclude this introduction to CTL with further remarks on SUS and its relationship to CTL. There is a logic related to SUS which was already studied by Rabin and which he denoted SωS. This is the logic interpreted

in the countably branching tree ω^* where, for each i, there is a separate successor relation $\mathrm{suc}_i(\cdot,\cdot)$. Observe that—as noted in [60]—in this logic one cannot even express that all successors of the root belong to a certain set, which can easily be expressed in CTL and SUS.

Observe, too, that in SUS one can express that every vertex of a tree has at least two successors, namely by

$$\forall x(\exists y_0 \exists y_1(\mathrm{suc}(x,y_0) \wedge \mathrm{suc}(x,y_1) \wedge \neg y_0 = y_1).$$

This is, however, impossible in CTL. More precisely, CTL cannot distinguish between bisimilar transition systems whereas SUS can do this easily.

5.2 From CTL to nondeterministic tree automata

We next show how to arrive at good complexity bounds for satisfiability and model checking by following a refined automata-theoretic approach. For satisfiability, we can use nondeterministic automata and vary the approach we used for handling LTL in Section 4, while for model checking, we have to use alternating tree automata.

As pointed out above, the nondeterministic tree automaton model we defined in Section 3 was suited for binary trees only, which is not enough in the context of CTL. Here, we need an automaton model that can handle trees with arbitrary branching degree. We could use the tree automaton model explained in Section 5.1, but there is another model which is more appropriate. Following Janin and Walukiewicz [59], we use a tree automaton model which takes into account that properties like the one mentioned at the end of Section 5.1 cannot be expressed in CTL.

A generalized Büchi tree automaton in this context is a tuple

$$\mathscr{A} = (A, Q, Q_I, \Delta, \mathscr{F})$$

where A, Q, Q_I, and $\mathscr{F}$ are as with generalized Büchi (word) automata and $\Delta \subseteq Q \times A \times 2^Q \times 2^Q$ is a transition relation.

A transition of the form $(q, a, Q^{\mathsf{E}}, Q^{\mathsf{A}})$ is to be read as follows: If the automaton is in state q at vertex v and reads the label a, then it sends each state from Q^{E} to at least one of the successors of v and every successor of v is at least sent one of the states from $Q^{\mathsf{E}} \cup Q^{\mathsf{A}}$; the same successor can get sent several states.

Formally, a run of $\mathscr{A}$ on a tree $\mathscr{T}$ is a $(Q \times V^{\mathscr{T}})$-labeled tree $\mathscr{R}$ satisfying the following conditions.

(i) The root of $\mathscr{R}$ is labeled $(q, \mathrm{root}(\mathscr{T}))$ for some $q \in Q_I$.

(ii) For every vertex $w \in V^{\mathscr{R}}$, if (q, v) is the label of w, then there exists a transition $(q, l^{\mathscr{R}}(v), Q^{\mathsf{E}}, Q^{\mathsf{A}}) \in \Delta$ such that:

(a) For every $v' \in \mathrm{succs}^{\mathscr{T}}(v)$ there exists $w' \in \mathrm{succs}^{\mathscr{R}}(w)$ labeled (q', v') for some $q' \in Q^{\mathsf{E}} \cup Q^{\mathsf{A}}$, that is, every successor of v occurs in a label of a successor of w.

(b) For every $q' \in Q^{\mathsf{E}}$ there exist $v' \in \mathrm{succs}^{\mathscr{T}}(v)$ and $w' \in \mathrm{succs}^{\mathscr{R}}(w)$ such that w' is labeled (q', v). That is, every state from Q^{E} occurs at least once among all successors of w.

Such a run is accepting if every branch is accepting with respect to the given generalized Büchi condition just as this was defined for generalized Büchi word automata.

Observe that in this model the unlabeled tree underlying a run may not be the same as the unlabeled tree underlying a given input tree. Copies of subtrees may occur repeatedly.

As an example, let $P = \{p\}$ and $A = 2^P$ and consider the tree language L which contains all trees over A that satisfy the property that every branch is either finite or labeled $\{p\}$ infinitely often. An appropriate Büchi automaton has two states, $q_\varnothing$ and $q_{\{p\}}$, where $q_\varnothing$ is initial and $q_{\{p\}}$ is final, and the transitions are $(q, a, \{q_a\})$ and $(q, a, \varnothing, \varnothing)$ for any state q and letter a.

The idea for translating a given CTL formula into a nondeterministic tree automaton follows the translation of LTL into nondeterministic word automata: In each vertex, the automaton guesses which subformulas of the given formula are true and verifies this. The only difference is that the path quantifiers E and A are taken into account, which is technically somewhat involved. The details are given in Figure 23, where the following notation and terminology is used. Given a set Ψ of CTL formulas over a finite set P of propositional variables and a letter $a \in 2^P$ we say that Ψ is consistent with a if

- $\mathsf{ff} \notin \Psi$,
- $p \in \Psi$ iff $p \in a$, for all $p \in P$, and
- for $\psi \in \Psi$, if $\psi = \psi_0 \vee \psi_1$, then $\psi_i \in \Psi$ for some $i < 2$, and if $\psi = \psi_0 \wedge \psi_1$, then $\{\psi_0, \psi_1\} \subseteq \Psi$.

Further, a set Ψ' is a witness for $\mathsf{E}(\psi \mathsf{XU} \chi)$ if $\chi \in \Psi'$ or $\{\psi, \mathsf{E}(\psi \mathsf{XU} \chi)\} \subseteq \Psi'$. Similarly, Ψ' is a witness for $\mathsf{E}(\psi \mathsf{XR} \chi)$ if $\{\psi, \chi\} \subseteq \Psi'$ or $\{\chi, \mathsf{E}(\psi \mathsf{XR} \chi)\} \subseteq \Psi'$. The analogue terminology is used for A-formulas. When Ψ is a set of CTL formulas, then Ψ_{E} denotes the formulas of the form $\mathsf{E}(\psi \mathsf{XU} \chi)$ and $\mathsf{E}(\psi \mathsf{XR} \chi)$, that is, the set of all E-formulas in Ψ, and, similarly, Ψ_{A} denotes the set of all A-formulas in Ψ.

The only interesting aspect of the construction is (iv) of the definition of a transition. It would be more natural to omit (iv), and, indeed, the construction would then also be correct, but the resulting automaton would

Let P be a finite set of propositional variables and φ a CTL formula over P in positive normal form. The generalized Büchi tree automaton for φ with respect to P, denoted $\mathscr{A}[\varphi]$, is defined by

$$\mathscr{A}[\varphi] = (2^P, 2^{\mathrm{sub}(\varphi)}, Q_I, \Delta, \mathscr{F})$$

where $Q_I = \{\Psi \subseteq 2^{\mathrm{sub}(\varphi)} \colon \varphi \in \Psi\}$ and

$$\mathscr{F} = \{F_{\mathsf{Q}[\psi \mathsf{XU} \chi]} \colon \mathsf{Q}[\psi \,\mathsf{XU}\, \chi] \in \mathrm{sub}(\varphi) \text{ and } \mathsf{Q} \in \{\mathsf{E}, \mathsf{A}\}\}$$

with

$$F_{\mathsf{Q}[\psi \mathsf{XU} \chi]} = \{\Psi \subseteq \mathrm{sub}(\varphi) \colon \chi \in \Psi \text{ or } \mathsf{Q}[\psi \,\mathsf{XU}\, \chi] \notin \Psi\},$$

and where Δ contains a transition $(\Psi, a, Q^{\mathsf{E}}, Q^{\mathsf{A}})$ if the following conditions are satisfied:

(i) Ψ is consistent with a,

(ii) for every $\psi \in \Psi_{\mathsf{E}}$ there exists $\Psi' \in Q^{\mathsf{E}}$ which witnesses it and Q^{A} contains all $\Psi \subseteq \mathrm{sub}(\varphi)$ that contain a witness for every $\psi \in \Psi_{\mathsf{A}}$,

(iii) every $\Psi' \in Q^{\mathsf{E}}$ witnesses every $\psi \in \Psi_{\mathsf{A}}$,

(iv) $\left|Q^{\mathsf{E}}\right| \leq |\mathrm{sub}(\varphi)_{\mathsf{E}}|$.

FIGURE 23. From CTL to generalized Büchi tree automata

be too large. On the other hand, (iv) is not a real restriction, because the semantics of CTL requires only one "witness" for every existential path formula.

Before formally stating the correctness of the construction, we introduce a notion referring to the number of different states which can be assigned in a transition. We say that a nondeterministic tree automaton $\mathscr{A}$ is m-bounded if $\left|Q^{\mathsf{E}}\right| \leq m$ holds for every $(q, a, Q^{\mathsf{E}}, Q^{\mathsf{A}}) \in \Delta$.

Lemma 5.5. Let φ be an arbitrary CTL formula with n subformulas, m E-subformulas, and k U-subformulas. Then $\mathscr{A}[\varphi]$ is an $(m+1)$-bounded generalized Büchi tree automaton with 2^n states, k acceptance sets, and

Let $\mathscr{A}$ be a nondeterministic Büchi tree automaton. The emptiness game for $\mathscr{A}$, denoted $\mathscr{G}_\varnothing[\mathscr{A}]$, is defined by

$$\mathscr{G}_\varnothing[\mathscr{A}] = (Q, \Delta, q_I, M_0 \cup M_1, F)$$

where

$$M_0 = \{(q, Q^{\mathsf{E}}, Q^{\mathsf{A}}) \colon \exists a \exists Q^{\mathsf{A}}((q, a, Q^{\mathsf{E}}, Q^{\mathsf{A}}) \in \Delta)\}, \text{ and}$$
$$M_1 = \{(Q', q) \colon q \in Q'\}.$$

FIGURE 24. Emptiness game for nondeterministic Büchi tree automaton

such that $\mathscr{L}(\mathscr{A}[\varphi]) = \mathscr{L}(\varphi)$.

Proof sketch. The claim about the size of the automaton is trivial. The proof of its correctness can be carried out similar to the proof of Theorem 4.4, that is, one proves $\mathscr{L}(\mathscr{A}[\varphi]) \subseteq \mathscr{L}(\varphi)$ by induction on the structure of φ and $\mathscr{L}(\varphi) \subseteq \mathscr{L}(\mathscr{A}[\varphi])$ by constructing an accepting run directly. Q.E.D.

It is very easy to see that the construction from Figure 18 can also be used in this context to convert a generalized Büchi tree automaton into a Büchi automaton. To be more precise, an m-bounded generalized Büchi tree automaton with n states and k acceptance sets can be converted into an equivalent m-bounded Büchi tree automaton with $(k+1)n$ states.

So in order to solve the satisfiability problem for CTL we only need to solve the emptiness problem for Büchi tree automata in this context. There is a simple way to perform an emptiness test for nondeterministic tree automata, namely by using the same approach as for nondeterministic tree automata working on binary trees: The nonemptiness problem is phrased as a game. Given a nondeterministic Büchi tree automaton $\mathscr{A}$, we define a game which Player 0 wins if and only if some tree is accepted by $\mathscr{A}$. To this end, Player 0 tries to suggest suitable transitions while Player 1 tries to argue that Player 0's choices are not correct. The details of the construction are given in Figure 24.

Lemma 5.6. Let $\mathscr{A}$ be a nondeterministic Büchi tree automaton. Then the following are equivalent:

(A) $\mathscr{L}(\mathscr{A}) \neq \varnothing$.

(B) Player 0 wins $\mathscr{G}_{\varnothing}[\mathscr{A}]$.

Proof. The proof of the lemma can be carried out along the lines of the proof of Lemma 3.3. The only difference is due to the arbitrary branching degree, which can easily be taken care of. One only needs to observe that if there exists a tree which is accepted by $\mathscr{A}$, then there is a tree with branching degree at most $|Q|$ which is accepted. Q.E.D.

We have the following theorem:

Theorem 5.7 (Emerson-Halpern-Fischer-Ladner, [37, 45])**.** CTL satisfiability is complete for deterministic exponential time.

Proof. The decision procedure is as follows. A given CTL formula φ is first converted into an equivalent generalized Büchi tree automaton $\mathscr{A}$ using the construction from Figure 23. Then $\mathscr{A}$ is converted into an equivalent Büchi tree automaton $\mathscr{B}$ using the natural adaptation of the construction presented in Figure 18 to trees. In the third step, $\mathscr{B}$ is converted into the Büchi game $\mathscr{G}_{\varnothing}[\mathscr{B}]$, and, finally, the winner of this game is determined. (Recall that a Büchi condition is a parity condition with two different priorities.)

From Theorem 2.21 on the complexity of parity games it follows immediately that Büchi games (parity games with two different priorities) can be solved in polynomial time, which means we only need to show that the size of $\mathscr{G}_{\varnothing}[\mathscr{B}]$ is exponential in the size of the given formula φ and can be constructed in exponential time. The latter essentially amounts to showing that $\mathscr{B}$ is of exponential size.

Let n be the number of subformulas of φ. Then $\mathscr{A}$ is n-bounded with 2^n states and at most n acceptance sets. This means that the number of sets Q' occurring in the transitions of $\mathscr{A}$ is at most 2^{n^2}, so there are at most 2^{n^2+2n} transitions (recall that there are at most 2^n letters in the alphabet). Similarly, $\mathscr{B}$ is n-bounded, has at most $(n+1)2^n$ states, and $2^{O(n^2)}$ transitions.

The lower bound is given in [45]. Q.E.D.

5.3 From CTL to alternating tree automata

One of the crucial results of Emerson and Clarke on CTL is that model checking of CTL can be carried out in polynomial time. The decision procedure they suggested in [28] is a simple labeling algorithms. For every subformula ψ of a given formula φ they determine in which states of a given transition system $\mathscr{S}$ the formula ψ holds and in which it does not hold. This is trivial for atomic formulas. It is straightforward for conjunction and disjunction, provided it is known which of the conjuncts and disjuncts, respectively, hold. For XR- and XU-formulas, it amounts to simple graph searches.

Emerson and Clarke's procedure cannot easily be seen as a technique which could also be derived following an automata-theoretic approach. Consider the nondeterministic tree automaton we constructed in Figure 23. Its size is exponential in the size of the given formula (and this cannot be avoided), so it is unclear how using this automaton one can arrive at a polynomial-time procedure.

The key for developing an automata-theoretic approach, which is due to Kupferman, Vardi, and Wolper [71], is to use alternating tree automata similar to how we used alternating automata for LTL in Section 4 and to carefully analyze their structure.

An alternating Büchi tree automaton is a tuple

$$\mathscr{A} = (P, Q, q_I, \delta, F)$$

where P, Q, q_I, and F are as usual and δ is the transition function which assigns to each state a transition condition. The set of transition conditions over P and Q, denoted $\mathrm{TC}(P, Q)$, is the smallest set such that

(i) $\mathsf{tt}, \mathsf{ff} \in \mathrm{TC}(P, Q)$,

(ii) $p, \neg p \in \mathrm{TC}(P, Q)$ for every $p \in P$,

(iii) every positive boolean combination of states is in $\mathrm{TC}(P, Q)$,

(iv) $\Diamond\gamma, \Box\gamma \in \mathrm{TC}(P, Q)$ where γ is a positive boolean combination of states.

This definition is very similar to the definition for alternating automata on words. The main difference reflects that in a tree a "position" can have several successors: $\Diamond$ expresses that a copy of the automaton should be sent to one successor, while $\Box$ expresses that a copy of the automaton should be sent to all successors. So $\Diamond$ and $\Box$ are the two variants of $\bigcirc$.

There is another, minor difference: For tree automata, we allow positive boolean combinations of states in the scope of $\Diamond$ and $\Box$. We could have allowed this for word automata, too, but it would not have helped us. Here, it makes our constructions simpler, but the proofs will be slightly more involved.

Let $\mathscr{T}$ be a 2^P-labeled tree. A tree $\mathscr{R}$ with labels from $\mathrm{TC}(P, Q) \times V^{\mathscr{T}}$ is a run of $\mathscr{A}$ on $\mathscr{T}$ if $l^{\mathscr{R}}(\mathrm{root}(\mathscr{R})) = (q_I, \mathrm{root}(\mathscr{T}))$ and the following conditions are satisfied for every vertex $w \in V^{\mathscr{R}}$ with label (γ, v):

- $\gamma \neq \mathsf{ff}$,
- if $\gamma = p$, then $p \in l^{\mathscr{T}}(w)$, and if $\gamma = \neg p$, then $p \notin l^{\mathscr{T}}(w)$,

- if $\gamma = \Diamond\gamma'$, then there exists $v' \in \mathrm{sucs}^{\mathscr{T}}(v)$ and $w' \in \mathrm{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = (\gamma', v')$,
- if $\gamma = \Box\gamma'$, then for every $v' \in \mathrm{sucs}^{\mathscr{T}}(v)$ there exists $w' \in \mathrm{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = (\gamma', v')$,
- if $\gamma = \gamma_0 \vee \gamma_1$, then there exists $i < 2$ and $w' \in \mathrm{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = (\gamma_i, v)$,
- if $\gamma = \gamma_0 \wedge \gamma_1$, then for every $i < 2$ there exists $w' \in \mathrm{sucs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = (\gamma_i, v)$.

Such a run is accepting if on every infinite branch there exist infinitely many vertices w labeled with an element of F in the first component.

The example language from above can be recognized by an alternating Büchi automaton which is slightly more complicated than the nondeterministic automaton, because of the restrictive syntax for transition conditions. We use the same states as above and four further states, q, $q'_{\{p\}}$, $q_\perp$, and $q'_\perp$. The transition function is determined by

$$\begin{aligned} \delta(q_I) &= q_\perp \vee q, & & \\ \delta(q_{\{p\}}) &= q'_{\{p\}} \wedge (q_\perp \vee q), & \delta(q'_{\{p\}}) &= p, \\ \delta(q_\perp) &= \Box q'_\perp, & \delta(q'_\perp) &= \mathsf{ff}, \\ \delta(q) &= \Box(q_I \vee q_{\{p\}}). & & \end{aligned}$$

The state $q_\perp$ is used to check that the automaton is at a vertex without successor.

In analogy to the construction for LTL, we can now construct an alternating tree automaton for a given CTL formula. This construction is depicted in Figure 25.

Compared to the construction for LTL, there are the following minor differences. First, the definition of the transition function is no longer inductive, because we allow positive boolean combinations in the transition function. Second, we have positive boolean combinations of states in the scope of $\Diamond$ and $\Box$. This was not necessary with LTL, but it is necessary here. For instance, if we instead had $\delta([\mathsf{E}(\psi\mathsf{X}\mathsf{U}\chi)]) = \Diamond[\chi] \vee (\Diamond[\psi] \wedge \Diamond[\mathsf{E}(\psi\mathsf{X}\mathsf{U}\chi)])$, then this would clearly result in a false automaton because of the second disjunct.

We can make a similar observation as with the alternating automata that we constructed for LTL formulas. The automata are very weak in the sense that when we turn the subformula ordering into a linear ordering $\leq$ on the states, then for each state q, the transition conditions $\delta(q)$ contains only states q' such that $q \geq q'$.

Let φ be a CTL formula in positive normal form over P and Q the set which contains for each $\psi \in \mathrm{sub}(\varphi)$ an element denoted $[\psi]$. The automaton $\mathscr{A}^{\mathrm{alt}}[\varphi]$ is defined by

$$\mathscr{A}^{\mathrm{alt}}[\varphi] = (P, Q, [\varphi], \delta, F)$$

where

$$\begin{aligned} \delta([\mathsf{tt}]) &= \mathsf{tt}, & \delta([\mathsf{ff}]) &= \mathsf{ff}, \\ \delta([p]) &= p, & \delta([\neg p]) &= \neg p, \\ \delta([\psi \vee \chi]) &= [\varphi] \vee [\psi], & \delta([\psi \wedge \chi]) &= [\varphi] \wedge [\psi], \end{aligned}$$

$$\begin{aligned} \delta([\mathsf{E}(\psi\, \mathsf{XU}\, \chi)]) &= \Diamond([\chi] \vee ([\psi] \wedge [\mathsf{E}(\psi\, \mathsf{XU}\, \chi)])), \\ \delta([\mathsf{E}(\psi\, \mathsf{XR}\, \chi)]) &= \Diamond([\chi] \wedge ([\chi] \vee [\mathsf{E}(\psi\, \mathsf{XR}\, \chi)])), \\ \delta([\mathsf{A}(\psi\, \mathsf{XU}\, \chi)]) &= \Box([\chi] \vee ([\psi] \wedge [\mathsf{A}(\psi\, \mathsf{XU}\, \chi)])), \\ \delta([\mathsf{A}(\psi\, \mathsf{XR}\, \chi)]) &= \Box([\chi] \wedge ([\chi] \vee [\mathsf{A}(\psi\, \mathsf{XR}\, \chi)])), \end{aligned}$$

and F contains all the elements $[\psi]$ where ψ is not an XU-formula.

FIGURE 25. From CTL to alternating tree automata

Lemma 5.8 (Kupferman-Vardi-Wolper, [71])**.** Let φ be a CTL formula with n subformulas. The automaton $\mathscr{A}^{\mathrm{alt}}[\varphi]$ is a very weak alternating tree automaton with n states and such that $\mathscr{L}(\mathscr{A}^{\mathrm{alt}}[A]) = \mathscr{L}(\varphi)$.

Proof. The proof can follow the lines of the proof of Lemma 4.7. Since the automaton is very weak, a simple induction on the structure of the formula can be carried out, just as in the proof of Lemma 4.7. Branching makes the proof only technically more involved, no new ideas are necessary to carry it out. Q.E.D.

As pointed out above, it is not our goal to turn $\mathscr{A}^{\mathrm{alt}}[\varphi]$ into a nondeterministic automaton (although this is possible), because such a translation cannot be useful for solving the model checking problem. What we rather do is to define a product of an alternating automaton with a transition system, resulting in a game, in such a way that the winner of the product of $\mathscr{A}^{\mathrm{alt}}[\varphi]$ with some transition system $\mathscr{S}$ reflects whether φ holds true in a certain state s_I of $\mathscr{S}$.

The idea is that a position in this game is of the form (γ, s) where γ is a transition condition and s is a state of the transition system. The goal is to design the game in such a way that Player 0 wins the game starting from (q_I, s_I) if and only if there exists an accepting run of the automaton on the unraveling of the transition system starting at s_I. This means, for instance, that if γ is a disjunction, then we make the position (γ, s) a position for Player 0, because by moving to one of the two successor positions he should show which of the disjuncts holds. If, on the other hand, $\gamma = \Box\gamma'$, then we make the position a position for Player 1, because she should be able to challenge Player 0 with any successor of s. The details are spelled out in Figure 26, where the following notation and terminology is used. Given an alternating automaton $\mathscr{A}$, we write $\mathrm{sub}(\mathscr{A})$ for the set of subformulas of the values of the transition function of $\mathscr{A}$. In addition, we write $\mathrm{sub}^+(\mathscr{A})$ for the set of all $\gamma \in \mathrm{sub}(\mathscr{A})$ where the maximum state occurring belongs to the set of final states.

Assume $\mathscr{A}$ is a very weak alternating Büchi automaton. Then $\mathscr{A} \times_{s_I} \mathscr{S}$ is not very weak in general in the sense that the game graph can be extended to a linear ordering. Observe, however, that the following is true for every position (q, s): All states in the strongly connected component of (q, s) are of the form (γ, s') where q is the largest state occurring in γ. So, by definition of $\mathscr{A} \times_{s_I} \mathscr{S}$, all positions in a strongly connected component of $\mathscr{A} \times_{s_I} \mathscr{S}$ are either final or nonfinal. We turn this into a definition. We say that a Büchi game is weak if for every strongly connected component of the game graph it is true that either all its positions are final or none of them is.

Lemma 5.9. Let $\mathscr{A}$ be an alternating Büchi tree automaton, $\mathscr{S}$ a transition system over the same finite set of propositional variables, and $s_I \in S$. Then $\mathscr{T}_{s_I}(\mathscr{S}) \in \mathscr{L}(\mathscr{A})$ iff Player 0 wins $\mathscr{A} \times_{s_I} \mathscr{S}$. Moreover, if $\mathscr{A}$ is a very weak alternating automaton, then $\mathscr{A} \times_{s_I} \mathscr{S}$ is a weak game.

Proof. The additional claim is obvious. For the other claim, first assume $\mathscr{R}$ is an accepting run of $\mathscr{A}$ on $\mathscr{T}_{s_I}(\mathscr{S})$. We convert $\mathscr{R}$ into a winning strategy σ for Player 0 in $\mathscr{A} \times \mathscr{S}$. To this end, let w be a vertex of $\mathscr{R}$ with label (γ, v) such that (γ, v) is a position for Player 0. Since $\mathscr{R}$ is an accepting run, w has a successor, say w'. Assume $l^{\mathscr{R}}(w') = (\gamma', v')$. We set $\sigma(u) = (\gamma, v'(*))$ where u is defined as follows. First, let $n = |v|$. Assume $l^{\mathscr{R}}(u(i)) = (\gamma_i, v_i)$ for every $i < n$. We set $u = (\gamma_0, v_0(*))(\gamma_1, v_1(*)) \dots (\gamma_{n-1}, v_{n-1}(*))$. It can be shown that this defines a strategy. Moreover, since $\mathscr{R}$ is accepting, σ is winning.

For the other direction, a winning strategy is turned into an accepting run in a similar manner. Q.E.D.

The proof shows that essentially there is no difference between a run and a strategy—one can think of a run as a strategy. From this point of

Let $\mathscr{A}$ be an alternating Büchi tree automaton, $\mathscr{S}$ a transition system over the same set of propositional variables, and $s_I \in S$. The product of $\mathscr{A}$ and $\mathscr{S}$ at s_I, denoted $\mathscr{A} \times_{s_I} \mathscr{S}$, is the Büchi game defined by

$$\mathscr{A} \times_{s_I} \mathscr{S} = (P_0, P_1, (q_I, s_I), M, \mathrm{sub}^+(\mathscr{A}) \times S)$$

where

- P_0 is the set of pairs $(\gamma, s) \in \mathrm{sub}(\mathscr{A}) \times S$ where γ is
  - (i) a disjunction,
  - (ii) a $\Diamond$-formula,
  - (iii) p for $p \notin l(s)$,
  - (iv) $\neg p$ for $p \in l(s)$, or
  - (v) ff,

  and
- P_1 is the set of pairs $(\gamma, s) \in \mathrm{sub}(\mathscr{A}) \times S$ where γ is
  - (i) a conjunction,
  - (ii) a $\Box$-formula,
  - (iii) p for some $p \in l(s)$,
  - (iv) $\neg p$ for some $p \notin l(s)$, or
  - (v) tt.

Further, M contains for every $\gamma \in \mathrm{sub}(\mathscr{A})$ and every $s \in S$ moves according to the following rules:

- if $\gamma = q$ for some state q, then $((\gamma, s), (\delta(q), s)) \in M$,
- if $\gamma = \gamma_0 \vee \gamma_1$ or $\gamma = \gamma_0 \wedge \gamma_1$, then $((\gamma, s), (\gamma_i, s)) \in M$ for $i < 2$,
- if $\gamma = \Diamond\gamma'$ or $\gamma = \Box\gamma'$, then $((\gamma, s), (\gamma', s')) \in M$ for all $s' \in \mathrm{succ}^{\mathscr{S}}(s)$.

FIGURE 26. Product of a transition system and an alternating automaton

view, an alternating automaton defines a family of games, for each tree a separate game, and the tree language recognized by the tree automaton is the set of all trees which Player 0 wins the game for.

The additional claim in the above lemma allows us to prove the desired complexity bound for the CTL model checking problem:

Theorem 5.10 (Clarke-Emerson-Sistla, [28])**.** The CTL model checking problem can be solved in time $O(mn)$ where m is the size of the transition system and n the number of subformulas of the CTL formula.

Proof. Consider the following algorithm, given a CTL formula φ, a transition system $\mathscr{S}$, and a state $s_I \in S$. First, construct the very weak alternating Büchi automaton $\mathscr{A}^{\mathrm{alt}}[\varphi]$. Second, build the product $\mathscr{A}^{\mathrm{alt}}[\varphi] \times_{s_I} \mathscr{S}$. Third, solve $\mathscr{A}^{\mathrm{alt}}[\varphi] \times_{s_I} \mathscr{S}$. Then Player 0 is the winner if and only if $\mathscr{S}, s_I \models \varphi$.

The claim about the complexity follows from the fact that the size of $\mathscr{A}^{\mathrm{alt}}[\varphi] \times_{s_I} \mathscr{S}$ is mn and from Theorem 2.21. Note that weak games are parity games with one priority in each strongly connected component. Q.E.D.

Obviously, given a CTL formula φ, a transition system $\mathscr{S}$, and a state s_I one can directly construct a game that reflects whether $\mathscr{S}, s_I \models \varphi$. This game would be called the model checking game for $\mathscr{S}$, s_I, and φ. The construction via the alternating automaton has the advantage that starting from this automaton one can solve both, model checking and satisfiability, the latter by using a translation from alternating Büchi tree automata into nondeterministic tree automata. We present such a translation in Section 6.

The translation from CTL into very weak alternating automata has another interesting feature. Just as the translation from LTL to weak alternating automata, it has a converse. More precisely, following the lines of the proof of Theorem 4.9, one can prove:

Theorem 5.11. Every very weak alternating tree automaton is equivalent to a CTL formula. Q.E.D.

5.4 Notes

The two specification logics that we have dealt with, LTL and CTL, can easily be combined into a single specification logic. This led Emerson and Halpern to introduce CTL* in 1986 [38].

An automata-theoretic proof of Corollary 5.7 was given first by Vardi and Wolper in 1986 [125]. Kupferman, Vardi, and Wolper, when proposing an automata-theoretic approach to CTL model checking in [71], also showed how other model checking problems can be solved following the automata-theoretic paradigm. One of their results is that CTL model checking can be solved in space polylogarithmic in the size of the transition system.

6 Modal μ-calculus

The logics that have been discussed thus far—S1S, S2S, LTL, and CTL—could be termed declarative in the sense that they are used to *describe* properties of sequences, trees, or transition systems rather than to specify how it can be determined whether such properties hold. This is different for the logic we discuss in this section, the modal μ-calculus (MC), introduced by Kozen in 1983 [66]. This calculus has a rich and deep mathematical and algorithmic theory, which has been developed over more than 20 years. Fundamental work on it has been carried out by Emerson, Streett, and Jutla [114, 40], Walukiewicz [129], Bradfield and Lenzi [79, 11], and others, and it has been treated extensively in books, for instance, by Arnold and Niwiński [6] and Stirling [110]. In this section, we study satisfiability (and model checking) for MC from an automata-theoretic perspective. Given that MC is much more complex than LTL or CTL, our exposition is less detailed, but gives a good impression of how the automata-theoretic paradigm works for MC.

6.1 MC and monadic second-order logic

MC is a formal language consisting of expressions which are evaluated in transition systems; every closed expression (without free variables) is evaluated to a set of states. The operations available for composing sets of states are boolean operations, local operations, and fixed point operations.

Formally, the set of MC expressions is the smallest set containing

- p and $\neg p$ for any propositional variable p,
- any fixed-point variable X,
- $\varphi \wedge \psi$ and $\varphi \vee \psi$ if φ and ψ are MC expressions,
- $\langle\rangle\varphi$ and $[]\varphi$ if φ is an MC expression, and
- $\mu X\varphi$ and $\nu X\varphi$ if X is a fixed-point variable and φ an MC expression.

The operators μ and ν are viewed as quantifiers in the sense that one says they bind the following variable. As usual, an expression without free occurrences of variables is called closed. The set of all variables occurring free in an MC expression φ is denoted by $\mathrm{free}(\varphi)$. An expression is called a fixed-point expression if it starts with μ or ν.

To define the semantics of MC expressions, let φ be an MC expression over some finite set P of propositional variables, $\mathscr{S}$ a transition system, and α a variable assignment which assigns to every fixed-point variable a set of states of $\mathscr{S}$. The value of φ with respect to $\mathscr{S}$ and α, denoted $||\varphi||^{\alpha}_{\mathscr{S}}$,

is defined as follows. The fixed-point variables and the propositional variables are interpreted according to the variable assignment and the transition system:

$$||p||^{\alpha}_{\mathscr{S}} = \{s \in S^{\mathscr{S}} : p \in l^{\mathscr{S}}(s)\}, \qquad ||\neg p||^{\alpha}_{\mathscr{S}} = \{s \in S^{\mathscr{S}} : p \notin l^{\mathscr{S}}(s)\},$$

and

$$||X||^{\alpha}_{\mathscr{S}} = \alpha(X).$$

Conjunction and disjunction are translated into union and intersection:

$$||\varphi \wedge \psi||^{\alpha}_{\mathscr{S}} = ||\varphi||^{\alpha}_{\mathscr{S}} \cap ||\psi||^{\alpha}_{\mathscr{S}}, \qquad ||\varphi \vee \psi||^{\alpha}_{\mathscr{S}} = ||\varphi||^{\alpha}_{\mathscr{S}} \cup ||\psi||^{\alpha}_{\mathscr{S}}.$$

The two local operators, $\langle\,\rangle$ and $[\,]$, are translated into graph-theoretic operations:

$$||\langle\,\rangle\varphi||^{\alpha}_{\mathscr{S}} = \{s \in S \colon \mathrm{succ}^{\mathscr{S}}(s) \cap ||\varphi||^{\alpha}_{\mathscr{S}} \neq \varnothing\},$$
$$||[\,]\varphi||^{\alpha}_{\mathscr{S}} = \{s \in S \colon \mathrm{succ}^{\mathscr{S}}(s) \subseteq ||\varphi||^{\alpha}_{\mathscr{S}}\}.$$

The semantics of the fixed-point operators is based on the observation that for every expression φ, the function $S' \mapsto ||\varphi||^{\alpha[X \mapsto S']}_{\mathscr{S}}$ is a monotone function on 2^S with set inclusion as ordering, where $\alpha[X \mapsto S']$ denotes the variable assignment which coincides with α, except for the value of the variable X, which is S'. The Knaster–Tarski Theorem then guarantees that this function has a least and a greatest fixed point:

$$||\mu X\varphi||^{\alpha}_{\mathscr{S}} = \bigcap \left\{ S' \subseteq S \colon ||\varphi||^{\alpha[X \mapsto S']}_{\mathscr{S}} = S' \right\},$$
$$||\nu X\varphi||^{\alpha}_{\mathscr{S}} = \bigcup \left\{ S' \subseteq S \colon ||\varphi||^{\alpha[X \mapsto S']}_{\mathscr{S}} = S' \right\}.$$

In the first equation the last equality sign can be replaced by $\subseteq$, while in the second equation it can be replaced by $\supseteq$. The above equations are—contrary to what was said at the beginning of this section—declarative rather than operational, but this can easily be changed because of the Knaster–Tarski Theorem. For a given system $\mathscr{S}$, a variable assignment α, an MC expression φ, and a fixed-point variable X, consider the ordinal sequence $(S_\lambda)_\lambda$, called approximation sequence for $||\mu X\varphi||^{\alpha}_{\mathscr{S}}$, defined by

$$S_0 = \varnothing, \qquad S_{\lambda+1} = ||\varphi||^{\alpha[X \mapsto S_\lambda]}_{\mathscr{S}}, \qquad S_{\lambda'} = \bigcup_{\lambda < \lambda'} S_\lambda,$$

where λ' stands for a limit ordinal. Because of monotonicity, we have $S_0 \subseteq S_1 \subseteq \ldots$. The definition of the sequence implies that if $S_\lambda = S_{\lambda+1}$ for any λ,

then $S_{\lambda'} = S_\lambda = ||\mu X\varphi||^\alpha_{\mathscr{S}}$ for all $\lambda' \geq \lambda$. Clearly, we have $\lambda \leq \mathrm{card}(S)$ for the smallest such λ, which, for finite transition systems, means there is a simple (recursive) way to evaluate $\mu X\varphi$. The same holds true for $\nu X\varphi$, where the approximation is from above, that is, $S_0 = S$ and the inclusion order is reversed.

For notational convenience, we also use $\mathscr{S}, \alpha, s \models \varphi$ to denote $s \in ||\varphi||^\alpha_{\mathscr{S}}$ for any state $s \in S$. When φ is a closed MC expression, then the variable assignment α is irrelevant for its interpretation, so we omit it and simply write $||\varphi||_{\mathscr{S}}$ or $\mathscr{S}, s \models \varphi$.

For examples of useful expressions, recall the CTL formula (1.3) from Section 5.1. We can express its subformula $\mathsf{EF}p_d$ by

$$\varphi_{\mathrm{inner}} = \mu X(p_d \vee \langle\rangle X),$$

so that the full formula can be written as

$$\nu Y(\varphi_{\mathrm{inner}} \wedge [\,]Y).$$

In a similar fashion, (1.4) can be expressed:

$$\nu Y((\neg p_r \vee \mu X(p_a \vee [\,]X)) \wedge [\,]Y).$$

It is more complicated to express the LTL formula (1.5); it needs a nested fixed-point expression with mutually dependent fixed-point variables. We first build an expression which denotes all states from which on all paths a state is reachable where p is true and which belongs to a set Y:

$$\varphi'_{\mathrm{inner}} = \mu X((p \wedge Y) \vee [\,]X).$$

Observe that Y occurs free in $\varphi'_{\mathrm{inner}}$. The desired expression can then be phrased as a greatest fixed point:

$$\nu Y \varphi'_{\mathrm{inner}}.$$

It is no coincidence that we are able to express the two CTL formulas in MC:

Proposition 6.1. For every CTL formula φ there exists a closed MC expression $\tilde{\varphi}$ such that for every transition system $\mathscr{S}$ and $s \in S$,

$$\mathscr{S}, s \models \varphi \qquad \text{iff} \qquad \mathscr{S}, s \models \tilde{\varphi}.$$

Proof. The proof is a straightforward induction. We describe one case of the inductive step. Assume ψ and χ are CTL formulas and $\tilde{\psi}$ and $\tilde{\chi}$ are MC expressions such that the claim holds. We consider $\varphi = \mathsf{E}(\psi\,\mathsf{XU}\,\chi)$ and

want to construct $\tilde{\varphi}$ as desired. We simply express the semantics of φ by a fixed-point computation:

$$\tilde{\varphi} = \langle\rangle \mu X(\tilde{\chi} \vee (\tilde{\psi} \wedge \langle\rangle \tilde{X})).$$

The other cases can be dealt with in the same fashion. Q.E.D.

The next observation is that as far as satisfiability is concerned, we can restrict our considerations to trees, just as with CTL (recall Lemma 5.1).

Lemma 6.2. For every MC expression φ, transition system $\mathscr{S}$, variable assignment α, and state $s \in S$,

$$\mathscr{S}, \alpha, s \models \varphi \qquad \text{iff} \qquad \mathscr{T}_s(\mathscr{S}), \alpha \models \varphi.$$

(Recall that when we view a tree as a transition system, then we interpret formulas in the root of the tree unless stated otherwise.)

Proof. This can be proved by a straightforward induction on the structure of φ, using the following inductive claim:

$$\{v \in V^{\mathscr{T}_s(\mathscr{S})} \colon \mathscr{S}, \alpha, v(*) \models \varphi\} = ||\varphi||^{\alpha}_{\mathscr{T}_s(\mathscr{S})}.$$

This simply says that with regard to MC, there is no difference between a state s' in a given transition system $\mathscr{S}$ and every vertex v with $v(*) = s'$ in the unraveling of $\mathscr{S}$. Q.E.D.

Just as with CTL, the lemma allows us to work henceforth in the tree framework. For a closed MC expression φ with propositional variables from a set $P = \{p_0, \ldots, p_{n-1}\}$, the tree language defined by φ, denoted $\mathscr{L}(\varphi)$, is the set of all trees $\mathscr{T}$ over 2^P such that $\mathscr{T} \models \varphi$.

The next observation is that every MC expression can be translated into a monadic second-order formula, similar to Proposition 5.2. Before we can state the result, we define an appropriate equivalence relation between SUS formulas and MC expressions. Recall that an SUS formula is true or not for a given tree, while an MC expression evaluates to a set of vertices.

Let $P = \{p_0, \ldots, p_{n-1}\}$ be a set of propositional variables and φ an MC expression over P with free fixed-point variables among $X_0, \ldots, X_{m-1}$. We view the variables $X_0, \ldots, X_{m-1}$ as further propositional variables and identify each X_i with a set variable V_i and each p_j with a set variable V_{m+j}. So we can interpret φ and every SUS formula $\psi = \psi(V_0, \ldots, V_{m+n-1})$ in trees over $[2]_{m+n}$. We say φ is equivalent to such a formula ψ if $\mathscr{L}(\varphi) = \mathscr{L}(\psi)$.

Proposition 6.3. For every MC expression φ, an equivalent SUS formula $\tilde{\varphi}$ can be constructed.

Proof. This can be proved by induction on the structure of φ, using a more general claim. For every MC expression φ as above, we construct an SUS formula $\hat{\varphi} = \hat{\varphi}(V_0, \ldots, V_{m+n-1}, z)$ such that for every tree $\mathscr{T}$ over $[2]_{m+n}$ and $v \in V^{\mathscr{T}}$, we have:

$$\mathscr{T}{\downarrow}v \models \varphi \qquad \text{iff} \qquad \mathscr{T}, v \models \hat{\varphi}(V_0, \ldots, V_{m+n-1}, z),$$

where $\mathscr{T}, v \models \hat{\varphi}(V_0, \ldots, V_{m+n-1}, z)$ is defined in the obvious way, see Section 4.1 for a similar definition in the context of LTL. We can then set $\tilde{\varphi} = \exists x(\forall y(\neg \mathrm{suc}(y, x) \wedge \hat{\varphi}(V_0, \ldots, V_{m+n-1}, x)))$.

The interesting cases in the inductive step are the fixed-point operators. So let $\varphi = \mu X_i \psi$ and assume $\hat{\psi}$ is already given. The formula $\hat{\varphi}$ simply says that z belongs to a fixed point and that every other fixed point is a superset of it:

$$\begin{aligned}\hat{\varphi} = \exists Z(z \in Z \wedge \forall z'(\hat{\psi}(\ldots, V_{i-1}, Z, V_{i+1}, \ldots, z') \leftrightarrow z' \in Z) \wedge \\ \forall Z'(\forall z'(\psi(\ldots, V_{i-1}, Z', V_{i+1}, \ldots, z') \leftrightarrow z' \in Z') \rightarrow Z \subseteq Z')).\end{aligned}$$

For the greatest fixed-point operator, the construction is analogous. Q.E.D.

As a consequence, we can state:

Corollary 6.4 (Kozen-Parikh, [67])**.** MC satisfiability is decidable.

But, just as with LTL and CTL, by a translation into monadic second-order logic we get only a nonelementary upper bound for the complexity.

6.2 From MC to alternating tree automata

Our overall objective is to derive a good upper bound for the complexity of MC satisfiability. The key is a translation of MC expressions into nondeterministic tree automata via alternating parity tree automata. We start with the translation of MC expressions into alternating parity tree automata.

Alternating parity tree automata are defined exactly as nondeterministic Büchi tree automata are defined in Section 5.3 except that the Büchi acceptance condition is replaced by a parity condition π.

Just as with LTL and CTL, the translation into alternating automata reflects the semantics of the expressions in a direct fashion. The fixed-point operators lead to loops, which means that the resulting tree automata will no longer be very weak (not even weak). For least fixed points these loops may not be traversed infinitely often, while this is necessary for greatest fixed points. To control this, priorities are used: Even priorities are used for greatest fixed-points, odd priorities for least fixed points. Different priorities are used to take into account the nesting of fixed points, the general rule being that outer fixed points have smaller priorities, because they are more important.

For model checking, it will be important to make sure as few different priorities as possible are used. That is why a careful definition of alternation depth is needed. In the approach by Emerson and Lei [41], one counts the number of alternations of least and greatest fixed points on the paths of the parse tree of a given expression. Niwiński's approach [92] yields a coarser hierarchy, which gives better upper bounds for model checking. It requires that relevant nested subexpressions are "mutually recursive".

Let $\leq$ denote the relation "is subexpression of", that is, $\psi \leq \varphi$ if $\psi \in \mathrm{sub}(\varphi)$. Let φ be an MC expression. An *alternating μ-chain* in φ of length l is a sequence

$$\varphi \geq \mu X_0 \psi_0 > \nu X_1 \psi_1 > \mu X_2 \psi_2 > \cdots > \mu/\nu X_{l-1} \psi_{l-1} \tag{1.6}$$

where, for every $i < l-1$, the variable X_i occurs free in every formula ψ with $\psi_i \geq \psi \geq \psi_{i+1}$. The maximum length of an alternating μ-chain in φ is denoted by $m^\mu(\varphi)$. Symmetrically, ν-chains and $m^\nu(\varphi)$ are defined. The *alternation depth* of a μ-calculus expression φ is the maximum of $m^\mu(\varphi)$ and $m^\nu(\varphi)$ and is denoted by $\mathrm{d}(\varphi)$.

We say an MC expression is in normal form if for every fixed-point variable X occurring the following holds:

- every occurrence of X in φ is free or
- all occurrences of X in φ are bound in the same subexpression $\mu X\psi$ or $\nu X\psi$, which is then denoted by φ_X.

Clearly, every MC expression is equivalent to an MC expression in normal form.

The full translation from MC into alternating parity tree automata can be found in Figure 27, where the following notation is used. When φ is an MC expression and $\mu X\psi \in \mathrm{sub}(\varphi)$, then

$$\mathrm{d}_\varphi(\mu X\psi) = \begin{cases} \mathrm{d}(\varphi) + 1 - 2\lceil \mathrm{d}(\mu X\psi)/2 \rceil, & \text{if } \mathrm{d}(\varphi) \bmod 2 = 0, \\ \mathrm{d}(\varphi) - 2\lfloor \mathrm{d}(\mu X\psi)/2 \rfloor, & \text{otherwise.} \end{cases}$$

Similarly, when $\nu X\psi \in \mathrm{sub}(\varphi)$, then

$$\mathrm{d}_\varphi(\nu X\psi) = \begin{cases} \mathrm{d}(\varphi) - 2\lfloor \mathrm{d}(\nu X\psi)/2 \rfloor, & \text{if } \mathrm{d}(\varphi) \bmod 2 = 0, \\ \mathrm{d}(\varphi) + 1 - 2\lceil \mathrm{d}(\nu X\psi)/2 \rceil, & \text{otherwise.} \end{cases}$$

This definition reverses alternation depth so it can be used for defining the priorities in the alternating parity automaton for an MC expression. Recall that we want to assign priorities such that the higher the alternation depth the lower the priority and, at the same time, even priorities go to ν-formulas

Let φ be a closed MC expression in normal form and Q a set which contains for every $\psi \in \mathrm{sub}(\varphi)$ a state $[\psi]$. The alternating parity tree automaton for φ, denoted $\mathscr{A}[\varphi]$, is defined by

$$\mathscr{A}[\varphi] = (P, Q, \varphi, \delta, \pi)$$

where the transition function is given by

$$\begin{aligned} \delta([p]) &= p, & \delta([\neg p]) &= \neg p, \\ \delta([\psi \vee \chi]) &= [\psi] \vee [\chi], & \delta([\psi \wedge \chi]) &= [\psi] \wedge [\chi], \\ \delta([\langle\rangle\psi]) &= \Diamond[\psi], & \delta([[]\psi]) &= \Box[\psi], \\ \delta([\mu X\psi]) &= [\psi], & \delta([\nu X\psi]) &= [\psi], \\ \delta([X]) &= [\varphi_X], \end{aligned}$$

and where

$$\pi([\psi]) = \mathrm{d}_\varphi(\psi)$$

for every fixed-point expression $\psi \in \mathrm{sub}(\varphi)$.

FIGURE 27. From μ-calculus to alternating tree automata

and odd priorities to μ-formulas. This is exactly what the above definition achieves.

It is obvious that $\mathscr{A}[\varphi]$ will have $\mathrm{d}(\varphi) + 1$ different priorities in general, but from a complexity point of view, these cases are not harmful. To explain this, we introduce the notion of index of an alternating tree automaton. The transition graph of an alternating tree automaton $\mathscr{A}$ is the graph with vertex set Q and where (q, q') is an edge if q' occurs in $\delta(q)$. The index of $\mathscr{A}$ is the maximum number of different priorities in the strongly connected components of the transition graph of $\mathscr{A}$. Clearly, $\mathscr{A}[\varphi]$ has index $\mathrm{d}(\varphi)$.

Theorem 6.5 (Emerson-Jutla, [40])**.** Let φ be an MC expression in normal form with n subformulas. Then $\mathscr{A}[\varphi]$ is an alternating parity tree automaton with n states and index $\mathrm{d}(\varphi)$ such that $\mathscr{L}(\mathscr{A}[\varphi]) = \mathscr{L}(\varphi)$.

To be more precise, $\mathscr{A}[\varphi]$ may have $\mathrm{d}(\varphi) + 1$ different priorities, but in every strongly connected component of the transition graph of $\mathscr{A}[\varphi]$ there are at most $\mathrm{d}(\varphi)$ different priorities, see also Theorem 2.21.

Proof. The claims about the number of states and the index are obviously true. The proof of correctness is more involved than the corresponding proofs for LTL and CTL, because the automata which result from the translation are, in general, not weak.

The proof of the claim is by induction on the structure of φ. The base case is trivial and so are the cases in the inductive step except for the cases where fixed-point operators are involved. We consider the case where $\varphi = \mu X \psi$.

So assume $\varphi = \mu X \psi$ and $\mathscr{T} \models \varphi$. Let $f\colon 2^V \to 2^V$ be defined by $f(V') = ||\psi||_{\mathscr{T}}^{X \mapsto V'}$. Let $(V_\lambda)_\lambda$ be the sequence defined by $V_0 = \varnothing$, $V_{\lambda+1} = f(V_\lambda)$, and $V_{\lambda'} = \bigcup_{\lambda<\lambda'} V_\lambda$ for limit ordinals λ'. We know that f has a least fixed point, which is the value of φ in $\mathscr{T}$, and that there exists κ such that V_κ is the least fixed-point of f. We show by induction on λ that there exists an accepting run of $\mathscr{A}[\varphi]$ on $\mathscr{T}{\downarrow}v$ for every $v \in V_\lambda$. This is trivial when $\lambda = 0$ or when λ is a limit ordinal. When λ is a successor ordinal, say $\lambda = \lambda_0 + 1$, then $V_\lambda = f(V_{\lambda_0})$. Consider the automaton $\mathscr{A}[\psi]$ where X is viewed as a propositional variable. By the outer induction hypothesis, there exists an accepting run $\mathscr{R}$ of $\mathscr{A}[\psi]$ on $\mathscr{T}[X \mapsto V_{\lambda_0}]{\downarrow}v$, where $\mathscr{T}[X \mapsto V_{\lambda_0}]$ is the obvious tree over $2^{P \cup \{X\}}$. We can turn $\mathscr{R}$ into a prefix $\mathscr{R}'$ of a run of $\mathscr{A}[\varphi]$ on $\mathscr{T}{\downarrow}v$ by adding a new root labeled $([\varphi], v)$ to it. Observe that some of the leaves w of $\mathscr{R}'$ may be labeled (X, v') with $v' \in V_{\lambda_0}$. For each such v' there exists, by the inner induction hypothesis, an accepting run $\mathscr{R}_{v'}$ of $\mathscr{A}[\varphi]$ on $\mathscr{T}{\downarrow}v$. Replacing w by $\mathscr{R}_{v'}$ for every such leaf w yields a run $\hat{\mathscr{R}}$ of $\mathscr{A}[\varphi]$ on $\mathscr{T}{\downarrow}v$. We claim this run is accepting. To see this, observe that each infinite branch of $\hat{\mathscr{R}}$ is an infinite branch of $\mathscr{R}'$ or has an infinite path of $\mathscr{R}_{v'}$ for some v' as a suffix. In the latter case, the branch is accepting for a trivial reason, in the former case, the branch is accepting because the priorities in $\mathscr{A}[\psi]$ differ from the priorities in $\mathscr{A}[\varphi]$ by a fixed even number. This completes the inductive proof. Since, by assumption, the root of $\mathscr{T}$ belongs to V_κ, we obtain the desired result.

For the other direction, assume $\mathscr{T}$ is accepted by $\mathscr{A}[\varphi]$, say by a run $\mathscr{R}$. Let W be the set of all $w \in V^{\mathscr{R}}$ such that φ is the first component of $l^{\mathscr{R}}(w)$. Observe that because of the definition of the priority function π there can only be a finite number of elements from W on each branch of $\mathscr{R}$. This is because the priority function π is defined in a way such that if $\psi \in \mathrm{sub}(\varphi)$ is a fixed-point formula with $[\psi]$ in the strongly connected component of $[\varphi]$ in the transition graph of $\mathscr{A}[\varphi]$, then $\pi([\varphi]) \leq \pi([\psi])$.

Consider the sequence $(V_\lambda)_\lambda$ of subsets of $V^{\mathscr{R}}$ defined as follows:

- $V_0 = \varnothing$,
- $w \in V_{\lambda+1}$ if all proper descendants of w in $\mathscr{R}$ belong to $V_\lambda \cup V^{\mathscr{R}} \setminus W$, and

- $V_{\lambda'} = \bigcup_{\lambda<\lambda'} V_\lambda$ for every limit ordinal λ'.

Using the induction hypothesis, one can prove by induction on λ that for every $w \in V_\lambda$ the second component of its label belongs to $||\varphi||_{\mathscr{T}}$.

Since there are only a finite number of elements from W on each branch of V_λ, one can also show that $\mathrm{root}(\mathscr{R}) \in W$, which proves the claim. Q.E.D.

Before we turn to the conversion of alternating into nondeterministic parity tree automata, we discuss model checking MC expressions briefly. Model checking an MC expression, that is, evaluating it in a finite transition system is "trivial" in the sense that one can simply evaluate the expression according to its semantics, using approximation for evaluating fixed-point operators as explained in Section 6.1. Using the fact that fixed-points of the same type can be evaluated in parallel one arrives at an algorithm which is linear in the product of the size of the expression and the size of the system, but exponential in the depth of the alternation between least and greatest fixed points.

An alternative approach to model checking MC expressions is to proceed as with CTL. Given a finite transition system $\mathscr{S}$, an initial state $s_I \in S$, and an expression φ, one first constructs the alternating automaton $\mathscr{A}[\varphi]$, then the product game $\mathscr{A}[\varphi] \times_{s_I} \mathscr{S}$ (with a parity condition rather than a Büchi condition), and finally solves this game. (Of course, on can also directly construct the game.) As a consequence of the previous theorem and Theorem 2.21, one obtains:

Theorem 6.6 (Seidl-Jurdziński, [107, 62])**.** An MC expression of size l and alternation depth d can be evaluated in a finite transition system with m states and n transitions in time $O((lm + ln(lm)^{\lfloor d/2 \rfloor}))$. Q.E.D.

In fact, there is a close connection between MC model checking and solving parity games: The two problems are interreducible, which means all the remarks on the complexity of solving parity games at the end of Section 2.5 are equally valid for MC model checking.

The above theorem tells us something about AMC, the set of all MC expressions with alternation depth ≤ 1. These expressions can be evaluated in time linear in the product of the size of the transition system and the length of the formula, which was first proved by Cleaveland, Klein, and Steffen [29] in general and by Kupferman, Vardi, and Wolper using automata-theoretic techniques [71]. This yields a different proof of Theorem 5.10: The straightforward translation from CTL into the μ-calculus, see Proposition 6.1, yields alternation-free expressions of linear size. From a practical point, it is interesting to note that model checking tools indeed use the translation of CTL into AMC, see [84].

6.3 From alternating to nondeterministic tree automata

In view of Theorem 6.5, what we need to solve MC satisfiability is a translation of alternating tree automata into nondeterministic tree automata, because we already know how to decide emptiness for these automata. To be precise, we proved this only for Büchi acceptance conditions, see Figure 24, but this extends to parity tree automata in a straightforward manner.

One way of achieving a translation from alternating into nondeterministic automata is to proceed in two steps, where the intermediate result is an alternating automaton with very restrictive transition conditions. We say a transition condition is in normal form if it is a disjunction of transition conditions of the form

$$\bigwedge_{q \in Q^{\mathsf{A}}} \Box q \wedge \bigwedge_{q \in Q^{\mathsf{E}}} \Diamond q.$$

The conversion of an ordinary alternating tree automaton into an alternating tree automaton with transition conditions in normal form is similar to removing ε-transitions. We describe it here for the case where the transition conditions are simpler as in the general case, namely where each subformula $\Box\gamma$ or $\Diamond\gamma$ is such that γ is a state. Observe that all the transition conditions in the construction described in Figure 27 are of this form. At the same time, we change the format of the transition function slightly. We say an alternating automaton is in normal form if its transition function δ is of the form $\delta\colon Q \times 2^P \to \mathrm{TC}(P, Q)$ where $\delta(q, a)$ is a transition condition in normal form for $q \in Q$ and $a \in 2^P$. The notion of a run of an alternating automaton is adapted appropriately.

To convert alternating automata into normal form, we start with a crucial definition. Let $\mathscr{A}$ be an alternating parity tree automaton, $a \in 2^P$, and $q \in Q$. We say a tree $\mathscr{R}$ labeled with transition conditions is a transition tree for q and a if its root is labeled q and every vertex w with label γ satisfies the following conditions:

- if $\gamma = p$, then $p \in a$, and if $\gamma = \neg p$, then $p \notin a$,
- if $\gamma = q'$, then there exists $w' \in \mathrm{succs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = \delta(q')$,
- if $\gamma = \Diamond q'$ or $\gamma = \Box q'$, then w has no successor,
- if $\gamma = \gamma_0 \vee \gamma_1$, then there exists $i < 2$ and $w' \in \mathrm{succs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = \gamma_i$,
- if $\gamma = \gamma_0 \wedge \gamma_1$, then for every $i < 2$ there exists $w' \in \mathrm{succs}^{\mathscr{R}}(w)$ such that $l^{\mathscr{R}}(w') = \gamma_i$.

Let $\mathscr{A}$ be an alternating parity tree automaton. The normalization of $\mathscr{A}$ is the alternating parity tree automaton $\mathscr{A}^{\mathrm{norm}}$ defined by

$$\mathscr{A}^{\mathrm{norm}} = (P, Q \times \pi(Q), (q_I, j), \delta', \pi')$$

where

- j is any element of $\pi(Q)$,
- $\pi'((q,i)) = i$ for all $q \in Q$ and $i \in \pi(Q)$, and
- $\delta'((q,i),a) = \bigvee \gamma_{\mathscr{R}}$ for $q \in Q$, $i \in \pi(Q)$, and $a \in 2^P$, with $\mathscr{R}$ ranging over all transition trees for q and a.

FIGURE 28. Normalizing transition conditions of alternating tree automata

Further, every infinite branch of $\mathscr{R}$ is accepting with respect to π.

A transition tree as above can easily be turned into a transition condition in normal form over an extended set of states, namely $\bar{Q} = Q \times \pi(Q)$. The second component is used to remember the minimum priority seen on a path of a transition tree, as explained below. Let Q^{A} be the set of pairs (q', i) such that $\Box q'$ is a label of a leaf of $\mathscr{R}$, say w, and i is the minimum priority on the path from the root of $\mathscr{R}$ to w. Similarly, let Q^{E} be the set of pairs (q', i) such that $\Diamond q'$ is a label of a leaf of $\mathscr{R}$, say w, and i is the minimum priority on the path from the root of $\mathscr{R}$ to w. The transition condition for the transition tree $\mathscr{R}$, denoted $\gamma_{\mathscr{R}}$, is defined by

$$\gamma_{\mathscr{R}} = \bigwedge_{(q',i) \in Q^{\mathsf{A}}} \Box(q', i) \wedge \bigwedge_{(q',i) \in Q^{\mathsf{E}}} \Diamond(q', i).$$

The entire normalization construction is depicted in Figure 28.

Lemma 6.7. Let $\mathscr{A}$ be an alternating parity tree automaton with n states and k different priorities. Then $\mathscr{A}^{\mathrm{norm}}$ is an alternating parity tree automaton in normal form with kn states and k different priorities. Q.E.D.

The second step in our construction is a conversion of an alternating automaton in normal form into a nondeterministic tree automaton, similar to the conversion of universal parity tree automata into nondeterministic tree automata explained in Section 3.3. Again, we heavily draw on the

generic automaton introduced in that section. Recall that given a finite state set Q and a priority function π, the generic automaton is a deterministic automaton over $\mathscr{Q}$, the alphabet consisting of all binary relations over Q, which accepts a word $u \in \mathscr{Q}^\omega$ if all $v \in \langle u \rangle$ satisfy the parity condition π.

Given an alternating automaton in normal form, a set $Q' \subseteq Q$, and a letter $a \in 2^P$, a pair $(\mathscr{Q}', R)$ with $\mathscr{Q}' \subseteq \mathscr{Q}$ and $R \in \mathscr{Q}$ is a choice for Q' and a if for every $q \in Q'$ there exists a disjunct in $\delta(q)$ of the form

$$\bigwedge_{q' \in Q_q^{\mathsf{A}}} \Box q' \wedge \bigwedge_{q' \in Q_q^{\mathsf{E}}} \Diamond q'$$

such that the following conditions are satisfied:

(i) $R = \{(q, q') \colon q \in Q' \wedge q' \in Q_q^{\mathsf{A}}\}$,

(ii) $R \subseteq R'$ for every $R' \in \mathscr{Q}'$,

(iii) for every $q \in Q'$ and every $q' \in Q_q^{\mathsf{E}}$ there exists $R' \in \mathscr{Q}'$ such that $(q, q') \in R'$, and

(iv) $|\mathscr{Q}'| \leq |Q| \times |Q| + 1$.

For a set $Q' \subseteq Q$ and a relation $R \subseteq Q \times Q$, we write $Q'R$ for the set $\{q' \in Q \colon \exists q (q \in Q' \wedge (q, q') \in R\}$.

The details of the conversion from alternating parity tree automata in normal form into nondeterministic tree automata can be found in Figure 29. It is analogous to the construction depicted in Figure 16, which describes how a universal parity tree automaton over binary trees can be turned into a nondeterministic parity tree automaton. The situation for alternating automata is different in the sense that the transition conditions of the form $\Diamond q'$ have to be taken care of, too, but this is captured by (iii) in the above definition.

Lemma 6.8. Let $\mathscr{A}$ be an alternating parity automaton in normal form with n states and k different priorities. Then A^{nd} is an equivalent nondeterministic automaton with a number of states exponential in n and a number of priorities polynomial in n.

Proof. The claims about the number of states and number of priorities are obvious. The correctness proof can be carried out almost in the same fashion as the proof of Lemma 3.10, except for one issue. In order to see that it is admissible to merge all branches of a run on a certain branch of a given tree into one element of $\mathscr{Q}^\omega$, one has to use Theorem 2.20, the memoryless determinacy of parity games. Q.E.D.

Let $\mathscr{A}$ be an alternating parity tree automaton in normal form and $\mathscr{B} = \mathscr{A}[Q^{\mathscr{A}}, \pi^{\mathscr{A}}]$ the generic automaton for $Q^{\mathscr{A}}$ and $\pi^{\mathscr{A}}$.

The nondeterministic automaton $\mathscr{A}^{\mathrm{nd}}$ is defined by

$$\mathscr{A}^{\mathrm{nd}} = (2^P, 2^{Q^{\mathscr{A}}} \times Q^{\mathscr{B}}, (\{q_I^{\mathscr{A}}\}, q_I^{\mathscr{B}}), \Delta, \pi)$$

where $\pi((Q', q)) = \pi^{\mathscr{B}}(q)$ and $((Q', q), q, \bar{\mathscr{Q}}, \{(Q'R', \delta^{\mathscr{B}}(q, R'))\}) \in \Delta$ if there exists a choice $(\mathscr{Q}', R)$ for Q' and a such that

$$\bar{\mathscr{Q}} = \{(Q'R, \delta^{\mathscr{B}}(q, R))\colon R \in \mathscr{Q}'\}.$$

FIGURE 29. From alternating to nondeterministic tree automata

As a consequence of Theorem 6.5 and Lemmas 6.7 and 6.8, we obtain:

Corollary 6.9. (Emerson-Streett-Jutla, [40]) Every MC expression can be translated into an equivalent nondeterministic parity tree automaton with an exponential number of states and a polynomial number of different priorities.

In view of Lemma 5.6 and Theorem 2.21, we can also conclude:

Corollary 6.10 (Emerson-Jutla, [39])**.** MC satisfiability is complete for exponential time.

For the lower bound, we refer to [39]. We finally note that a converse of Corollary 6.9 also holds:

Theorem 6.11 (Niwiński-Emerson-Jutla-Janin-Walukiewicz, [93, 40, 59])**.** Let P be a finite set of propositional variables. For every alternating parity tree automaton and every nondeterministic tree automaton over 2^P, there exists an equivalent closed MC expression.

6.4 Notes

Satisfiability for MC is not only complexity-wise simpler than satisfiability for S2S. The proofs for showing decidability of satisfiability for S2S all make use of a determinization construction for automata on infinite words. The "safraless decision procedures" advocated by Kupferman and Vardi

[75] avoid this, but they still use the fact that equivalent deterministic word automata of a bounded size exist.

The nondeterministic tree automaton models for SUS and MC are not only similar on the surface: A fundamental result by Janin and Walukiewicz [60] states that the bisimulation-invariant tree languages definable in SUS are exactly the tree languages definable in MC, where the notion of bisimulation exactly captures the phenomenon that MC expressions (just as CTL formulas) are resistant against duplicating subtrees.

MC has been extended in various ways with many different objectives. With regard to adding to its expressive power while retaining decidability, one of the most interesting results is by Grädel and Walukiewicz [53], which says that satisfiability is decidable for guarded fixed-point logic. This logic can be seen as an extension of the modal μ-calculus insofar as guarded logic is considered a natural extension of modal logic, and guarded fixed-point logic is an extension of guarded logic just as modal μ-calculus is an extension of model logic by fixed-point operators. For further extensions, see [78, 127] and [68]. Other important work with regard to algorithmic handling of MC was carried out by Walukiewicz in [130], where he studies the evaluation of MC expressions on pushdown graphs.

References

[1] C. S. Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of büchi automata. *Theor. Comput. Sci.*, 363(2):224–233, 2006.

[2] R. Armoni, L. Fix, A. Flaisher, R. Gerth, B. Ginsburg, T. Kanza, A. Landver, S. Mador-Haim, E. Singerman, A. Tiemeyer, M. Vardi, and Y. Zbar. The ForSpec temporal logic: A new temporal property-specification logic. In J.-P. Katoen and P. Stevens, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 8th International Conference, TACAS 2002, Held as Part of the Joint European Conference on Theory and Practice of Software, ETAPS 2002, Grenoble, France, April 8-12, 2002, Proceedings*, volume 2280 of *Lecture Notes in Computer Science*, pages 296–211. Springer, 2002.

[3] R. Armoni, D. Korchemny, A. Tiemeyer, and M. Y. V. Y. Zbar. Deterministic dynamic monitors for linear-time assertions. In *Proc. Workshop on Formal Approaches to Testing and Runtime Verification*, volume 4262 of *Lecture Notes in Computer Science*. Springer, 2006.

[4] A. Arnold. Rational omega-languages are non-ambiguous. *Theor. Comput. Sci.*, 26:221–223, 1983.

[5] A. Arnold, J. Duparc, D. Niwiński, and F. Murlak. On the topological complexity of tree languages. This volume.

[6] A. Arnold and D. Niwiński. *Rudiments of μ-Calculus*. Elsevier, Amsterdam, The Netherlands, 2001.

[7] M. Ben-Ari, Z. Manna, and A. Pnueli. The logic of nexttime. In *Proc. 8th ACM Symp. on Principles of Programming Languages (POPL)*, pages 164–176, 1981.

[8] D. Berwanger, A. Dawar, P. Hunter, and S. Kreutzer. Dag-width and parity games. In B. Durand and W. Thomas, editors, *STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23–25, 2006, Proceedings*, volume 3884 of *Lecture Notes in Computer Science*, pages 524–536, 2006.

[9] A. Blumensath, T. Colcombet, and C. Löding. Logical theories and compatible operations. This volume.

[10] M. Bojanczyk. The common fragment of ctl and ltl needs existential modalities. Available as `http://www.mimuw.edu.pl/~bojan/papers/paradox.pdf`, 2007.

[11] J. C. Bradfield. The modal μ-calculus alternation hierarchy is strict. *Theor. Comput. Sci.*, 195(2):133–153, 1998.

[12] J. R. Büchi. Using determinancy of games to eliminate quantifiers. In *FCT*, pages 367–378, 1977.

[13] R. M. Burstall. Program proving as hand simulation with a little induction. In *Information Processing 74*, pages 308–312, Stockholm, Sweden, Aug. 1974. International Federation for Information Processing, North-Holland Pub. Co.

[14] D. Bustan, A. Flaisher, O. Grumberg, O. Kupferman, and M. Y. Vardi. Regular vacuity. In *Proc. 13th Conf. on Correct Hardware Design and Verification Methods*, volume 3725 of *Lecture Notes in Computer Science*, pages 191–206. Springer, 2005.

[15] E. Börger, E. Grädel, and Y. Gurevich. *The Classical Decision Problems*. Springer, 1997.

[16] J. R. Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

[17] J. R. Büchi. On a decision method in restricted second-order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Logic, Methodology, and Philosophy of Science: Proc. of the 1960 International Congress*, pages 1–11, Stanford, Calif., 1962. Stanford University Press.

[18] J. R. Büchi. Decision methods in the theory of ordinals. *Bull. Am. Math. Soc*, 71:767–770, 1965.

[19] J. R. Büchi and L. H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. Amer. Math. Soc.*, 138:295–311, 1969.

[20] A. Carayol and C. Löding. MSO on the infinite binary tree: Choice and order. In *CSL'07*, volume 4646 of *LNCS*, pages 161–176. Springer, 2007.

[21] O. Carton, D. Perrin, and J.-É. Pin. Automata and semigroups recognizing infinite words. This volume.

[22] D. Caucal. Deterministic graph grammars. This volume.

[23] A. Church. Application of recursive arithmetics to the problem of circuit analysis. In I. f. D. A. Communications Research Division, editor, *Summaries of Talks Presented at the Summer Institute for Symbolic Logic, Ithaca, Cornell University, July 1957*, pages 3–50, 1960.

[24] A. Church. Logic, arithmetics, and automata. In I. Mittag-Leffler, editor, *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35, 1963.

[25] A. Cimatti, M. Roveri, S. Semprini, and S. Tonetta. From psl to nba: A modular symbolic encoding. In *Proc. 6th Int'l Conf. on Formal Methods in Computer-Aided design*, 2006.

[26] E. M. Clarke and I. A. Draghicescu. Expressibility results for linear-time and branching-time logics. In *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30–June 3, 1988, Proceedings*, volume 354 of *Lecture Notes in Computer Science*, pages 428–437. Springer, 1988.

[27] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications: a practical approach. In ACM, editor, *Conference Record of the Tenth Annual ACM Symposium on Principles of Programming Languages*, pages 117–126, Austin, Texas, 1983. ACM Press.

[28] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, Apr. 1986.

[29] R. Cleaveland, M. Klein, and B. Steffen. Faster model checking for the modal μ-calculus. In G. von Bochmann and D. K. Probst, editors, *Computer Aided Verification, Fourth International Workshop, CAV '92, Montreal, Canada, June 29 – July 1, 1992, Proceedings*, volume 663 of *Lecture Notes in Computer Science*, pages 410–422. Springer, 1992.

[30] V. Diekert and P. Gastin. First-order definable languages. This volume.

[31] J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, Oct. 1970.

[32] M. Droste and P. Gastin. Weighted automata and weighted logics. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 513–525. Springer, 2005.

[33] W. Ebinger and A. Muscholl. Logical definability on infinite traces. In A. Lingas, R. Karlsson, and S. Carlsson, editors, *Automata, Languages and Programming: 20th International Colloquium*, volume 700 of *Lecture Notes in Computer Science*, pages 335–346, Lund, Sweden, 1993. EATCS, Springer.

[34] C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Springer, 2006.

[35] C. C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, Jan. 1961.

[36] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.

[37] E. A. Emerson and J. Y. Halpern. Decision procedures and expressiveness in the temporal logic of branching time. *J. Comput. System Sci.*, 30:1–24, 1985.

[38] E. A. Emerson and J. Y. Halpern. Sometimes and not never revisited: On branching versus linear time. *J. Assoc. Comput. Mach.*, 33(1):151–178, 1986.

[39] E. A. Emerson and C. S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *32nd Annual Symposium on Foundations of Computer Science*, pages 328–337, San Juan, Puerto Rico, 1991. IEEE.

[40] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *32nd Annual Symposium on Foundations of Computer Science*, pages 368–377, San Juan, Puerto Rico, October 1991. IEEE.

[41] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *1st IEEE Symposium on Symposium on Logic in Computer Science*, pages 267–278, Cambridge, Massachusetts, 16–18 June 1986. IEEE Computer Society.

[42] E. A. Emerson and A. P. Sistla. Deciding branching time logic. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, 1984, Washington, D.C., USA*, pages 14–24. ACM, 1984.

[43] E. A. Emerson and A. P. Sistla. Deciding full branching time logic. *Information and Control*, 61(3):175–201, 1984.

[44] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown automata. In F. Moller, editor, *Proc. 2nd Int. Workshop on Verification of Infinite States Systems*, 1997.

[45] M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *J. Comput. System Sci.*, 18:194–211, 1979.

[46] E. Friedgut, O. Kupferman, and M. Y. Vardi. Büchi complementation made tighter. *Int. J. Found. Comput. Sci.*, 17(4):851–868, 2006.

[47] D. M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Conference Record of the 12th ACM Symposium on Principles of Programming Languages*, pages 163–173, Las Vegas, Nev., 1980.

[48] P. Gastin and D. Oddoux. Fast LTL to Büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *Computer Aided Verification, 13th International Conference, CAV 2001, Paris, France, July 18-22, 2001, Proceedings*, volume 2102 of *Lecture Notes in Computer Science*, pages 53–65. Springer, 2001.

[49] 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, 1995.

[50] D. Giammarresi and A. Restivo. Matrix based complexity functions and recognizable picture languages. This volume.

[51] D. Giammarresi, A. Restivo, S. Seibert, and W. Thomas. Monadic second-order logic over rectangular pictures and recognizability by tiling systems. *Information and Computation*, 125(1):32–45, Feb. 1996.

[52] E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002.

[53] E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *LICS*, pages 45–54, 1999.

[54] Y. Gurevich and L. Harrington. Trees, automata, and games. In *14th ACM Symposium on the Theory of Computing*, pages 60–65, San Francisco, 1982. ACM Press.

[55] S. Gurumurthy, O. Kupferman, F. Somenzi, and M. Vardi. On complementing nondeterministic Büchi automata. In *Proc. 12th Conf. on Correct Hardware Design and Verification Methods*, volume 2860 of *Lecture Notes in Computer Science*, pages 96–110. Springer, 2003.

[56] J. G. Henriksen, J. L. Jensen, M. E. Jørgensen, N. Klarlund, R. Paige, T. Rauhe, and A. Sandholm. Mona: Monadic second-order logic in practice. In E. Brinksma, R. Cleaveland, K. G. Larsen, T. Margaria, and B. Steffen, editors, *TACAS: Tools and Algorithms for Construction and Analysis of Systems, First International Workshop, TACAS '95, Aarhus, Denmark, May 19–20, 1995, Proceedings*, volume 1019 of *Lecture Notes in Computer Science*, pages 89–110, 1995.

[57] G. J. Holzmann. The model checker SPIN. *IEEE Trans. Software Engrg.*, 23(5):279–295, 1997.

[58] N. Immerman and D. Kozen. Definability with bounded number of bound variables. *Information and Computation*, 83(2):121–139, Nov. 1989.

[59] D. Janin and I. Walukiewicz. Automata for the modal mu-calculus and related results. In J. Wiedermann and P. Hájek, editors, *Mathematical Foundations of Computer Science 1995, 20th International Symposium, MFCS'95, Prague, Czech Republic, August 28 – September 1, 1995, Proceedings (MFCS)*, pages 552–562, 1995.

[60] D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In U. Montanari and V. Sassone, editors, *CONCUR '96, Concurrency Theory, 7th International Conference, Pisa, Italy, August 26-29, 1996, Proceedings*, volume 1119 of *Lecture Notes in Computer Science*, pages 263–277, 1996.

[61] M. Jurdziński. Deciding the winner in parity games is in UP ∩ co-UP. *Information Processing Letters*, 68(3):119–124, November 1998.

[62] M. Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *STACS 2000, 17th Annual Symposium on Theoretical Aspects of Computer Science, Lille, France, February 2000, Proceedings*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000.

[63] M. Jurdziński, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms, SODA 2006*, pages 117–123. ACM/SIAM, 2006.

[64] J. A. W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, Calif., 1968.

[65] N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *32nd Annual Symposium on Foundations of Computer Science, 1–4 October 1991, San Juan, Puerto Rico*, pages 358–367, 1991.

[66] D. Kozen. Results on the propositional μ-calculus. *Theor. Comput. Sci.*, 27:333–354, 1983.

[67] D. Kozen and R. Parikh. A decision procedure for the propositional μ-calculus. In *Logics of Programs*, volume 164 of *Lecture Notes in Computer Science*, pages 313–325. Springer, 1984.

[68] S. Kreutzer and M. Lange. Non-regular fixed-point logics and games. This volume.

[69] F. Kröger. LAR: A logic of algorithmic reasoning. *Acta Informatica*, 8(3), August 1977.

[70] O. Kupferman, N. Piterman, and M. Y. Vardi. Model checking linear properties of prefix-recognizable systems. In *Proc 14th Int. Conf. on Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 371–385. Springer, 2002.

[71] O. Kupferman, M. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *J. Assoc. Comput. Mach.*, 47(2):312–360, 2000.

[72] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. In *ISTCS*, pages 147–158, 1997.

[73] O. Kupferman and M. Y. Vardi. The weakness of self-complementation. In *STACS*, pages 455–466, 1999.

[74] O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Trans. Comput. Logic*, 2(3):408–429, 2001.

[75] O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), 23-25 October 2005, Pittsburgh, PA, USA, Proceedings (FOCS)*, pages 531–542. IEEE Computer Society, 2005.

[76] D. Kähler. Determinisierung von É-Automaten. Diploma thesis, Institut für Informatik, Christian-Albrechts-Universität zu Kiel, 2001.

[77] L. Lamport. "Sometimes" is sometimes "not never" - on the temporal logic of programs. In *Proc. 7th ACM Symp. on Principles of Programming Languages (POPL)*, pages 174–185, 1980.

[78] M. Lange and C. Stirling. Model checking fixed point logic with chop. In M. Nielsen and U. Engberg, editors, *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 250–263. Springer, 2002.

[79] G. Lenzi. A hierarchy theorem for the μ-calculus. In F. M. auf der Heide and B. Monien, editors, *Automata, Languages and Programming, 23rd International Colloquium, ICALP96, Paderborn, Germany, 8-12 July 1996, Proceedings*, volume 1099 of *Lecture Notes in Computer Science*, pages 87–97. Springer, 1996.

[80] O. Lichtenstein and A. Pnueli. Checking that finite state concurrent programs satisfy their linear specification. In *Proc. 12th ACM Symp. on Principles of Programming Languages"*, pages 97–107, 1985.

[81] C. Löding and W. Thomas. Alternating automata and logics over infinite words. In *Proceedings of the IFIP International Conference*

on Theoretical Computer Science, IFIP TCS2000, volume 1872 of *Lecture Notes in Computer Science*, pages 521–535. Springer, 2000.

[82] D. A. Martin. Borel determinacy. *Ann. Math.*, 102:363–371, 1975.

[83] O. Matz and N. Schweikardt. Expressive power of monadic logics on words, trees, pictures, and graphs. This volume.

[84] K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

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

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

[87] S. Miyano and T. Hayashi. Alternating finite automata on omega-words. In *CAAP*, pages 195–210, 1984.

[88] A. W. Mostowski. Games with forbidden positions. Preprint 78, Uniwersytet Gdańsk, Instytyt Matematyki, 1991.

[89] D. E. Muller. Infinite sequences and finite machines. In *Proceedings of the 4th Annual IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 3–16, 1963.

[90] D. E. Muller and P. E. Schupp. Alternating automata on infinite objects, determinacy and rabin's theorem. In M. Nivat and D. Perrin, editors, *Automata on Infinite Words, Ecole de Printemps d'Informatique Théorique, Le MontDore, May 14–18, 1984*, volume 192 of *Lecture Notes in Computer Science*, pages 100–107. Springer, 1985.

[91] D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of rabin, mcnaughton and safra. *Theor. Comput. Sci.*, 141(1&2):69–107, 1995.

[92] D. Niwiński. On fixed point clones. In L. Kott, editor, *Automata, Languages and Programming: 13th International Colloquium*, volume 226 of *Lecture Notes in Computer Science*, pages 464–473, Rennes, France, 1986. Springer-Verlag, Berlin.

[93] D. Niwinski. Fixed points vs. infinite generation. In *Proceedings, Third Annual Symposium on Logic in Computer Science, 5–8 July 1988, Edinburgh, Scotland, UK (LICS)*, pages 402–409. IEEE Computer Society, 1988.

[94] D. Niwiński and I. Walukiewicz. Ambiguity problem for automata on infinite trees. Unpublished note.

[95] D. Perrin. Recent results on automata on infinite words. In L. Kott, editor, *13th Intern. Coll. on Automata, Languages and Programming*, volume 226 of *Lecture Notes in Computer Science*, pages 134–148, Rennes, France, 1986. Springer-Verlag, Berlin.

[96] D. Perrin and J.-É. Pin. *Infinite Words: Automata, Semigroups, Logic and Games*. Pure and Applied Mathematics. Elsevier, 2003.

[97] N. Piterman. From nondeterministic büchi and streett automata to deterministic parity automata. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*, pages 255–264. IEEE Computer Society, 2006.

[98] A. Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science*, pages 46–57, Rhode Island, Providence, 1977. IEEE.

[99] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *POPL*, pages 179–190, 1989.

[100] M. O. Rabin. Decidability of second-order theories and finite automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.

[101] M. O. Rabin. Weakly definable relations and special automata. In *Proc. Symp. Mathematical Logic and Foundation of Set Theory*, pages 1–23. North Holland, 1970.

[102] K. Reinhardt. The complexity of translating logic to finite automata. In *Automata, Logics, and Infinite Games*, pages 231–238, 2001.

[103] G. S. Rohde. *Alternating automata and the temporal logic of ordinals*. PhD thesis, University of Illinois at Urbana-Champaign, 1997.

[104] R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Institute of Science, 1992.

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

[106] P. Schnoebelen. The complexity of temporal logic model checking. In *Advances in Modal Logic*, pages 393–436. King's College Publications, 2002.

[107] H. Seidl. Fast and simple nested fixpoints. *Information Processing Letters*, 59(6):303–308, 1996.

[108] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. In *Proceedings of the Fourteenth Annual ACM Symposium on Theory of Computing, 5–7 May 1982, San Francisco, California, USA (STOC)*, pages 159–168. ACM, 1982.

[109] A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *J. Assoc. Comput. Mach.*, 32(3):733–749, 1985.

[110] C. Stirling. *Modal and temporal properties of processes*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[111] L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, Dept. of Electrical Engineering, MIT, Boston, Mass., 1974.

[112] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Fifth Annual ACM Symposium on Theory of Computation*, pages 1–9, Austin, Texas, 1973. ACM Press.

[113] R. S. Streett. Propositional dynamic logic of looping and converse. *Inform. Contr.*, 54:121–141, 1982.

[114] R. S. Streett and E. A. Emerson. The propositional mu-calculus is elementary. In J. Paredaens, editor, *Automata, Languages and Programming, 11th Colloquium, Antwerp, Belgium, July 16-20, 1984, Proceedings (ICALP)*, volume 172 of *Lecture Notes in Computer Science*, pages 465–472. Springer, 1984.

[115] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order arithmetic. *Mathematical System Theory*, 2(1):57–81, 1968.

[116] W. Thomas. Star-free regular sets of ω-sequences. *Inform. and Control*, 42:148–156, 1979.

[117] W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25:360–376, 1982.

[118] W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, pages 134–191. Elsevier, Amsterdam, 1990.

[119] W. Thomas. On logical definability of regular trace languages. In V. Diekert, editor, *Proceedings of a Workshop of the ESPRIT Basic Research Action No. 3166: Algebraic and Syntactic Methods in Computer Science (ASMICS)*, pages 172–182, Kochel am See, BRD, 1990. Bericht TUM-I901902, Technische Universität München.

[120] W. Thomas. Languages, automata and logic. In A. Salomaa and G. Rozenberg, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 389–455. Springer, Berlin, 1997.

[121] W. Thomas. Complementation of büchi automata revised. In J. Karhumäki, H. A. Maurer, G. Paun, and G. Rozenberg, editors, *Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa*, pages 109–120. Springer, 1999.

[122] B. A. Trakhtenbrot. Finite automata and the logic of one-place predicates. *Siberian Math. J.*, 3:103–131, 1962. (English translation in: AMS Transl. 59 (1966) 23–55.).

[123] M. Y. Vardi. Automata-theoretic model checking revisited. In *Proc. 7th Int'l Conf. on Verification, Model Checking, and Abstract Interpretation*, volume 4349 of *Lecture Notes in Computer Science*, pages 137–150. Springer, 2007.

[124] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In D. Kozen, editor, *First Annual IEEE Symposium on Logic in Computer Science*, pages 322–331, Cambridge, Mass., 16–18 June 1986. IEEE.

[125] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32(2):182–221, 1986.

[126] M. Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 15 Nov. 1994.

[127] M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In P. Gardner and N. Yoshida, editors, *CONCUR*, volume 3170 of *Lecture Notes in Computer Science*, pages 512–528. Springer, 2004.

[128] I. Walukiewicz. Monadic second order logic on tree-like structures. In C. Puech and R. Reischuk, editors, *STACS 96, 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, February 22–24, 1996, Proceedings*, volume 1046 of *Lecture Notes in Computer Science*, pages 401–413, 1996.

[129] I. Walukiewicz. Completeness of kozen's axiomatisation of the propositional μ-calculus. *Inf. Comput.*, 157(1-2):142–182, 2000.

[130] I. Walukiewicz. Pushdown processes: Games and model-checking. *Inf. Comput.*, 164(2):234–263, 2001.

[131] Q. Yan. Lower bounds for complementation of *mega*-automata via the full automata technique. In M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener, editors, *Automata, Languages and Programming, 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10–14, 2006, Proceedings, Part II*, volume 4052 of *Lecture Notes in Computer Science*, pages 589–600, 2006.

[132] L. D. Zuck. *Past Temporal Logic*. PhD thesis, The Weizmann Institute of Science, Rehovot, Israel, Aug. 1986.