1 Determinization

Given an ω-regular language, it is often very useful to have a *deterministic* automaton that recognizes it, e.g., in synthesis we construct the game between the system and environment from a deterministic automaton that recognizes the winning plays of the system player. Unfortunately, deterministic Büchi automata only recognize a strict subset of the class of ω-regular languages. In this chapter, we resolve this shortcoming by introducing parity automata and Muller automata, whose respective acceptance conditions are more expressive.[1] We show that deterministic parity and deterministic Muller automata recognize precisely the class of ω-regular languages. We observe that the complementation of these deterministic automata to be a straightforward operation, and establish as a corollary that the class of ω-regular languages is closed under complementation.

1.1 Intuition: Determinization via Subset Tracking

We begin by recalling the classic subset tracking method to determinize finite-word automata: given a finite-word automaton $\mathcal{N} = (\Sigma, Q, I, T, F)$, we construct a deterministic finite-word automaton $\mathcal{D} = (\Sigma, 2^Q, I, T', \mathcal{F})$ that uses its states to track the subset of states that $\mathcal{N}$ could possibly be in, and accepts if the final state of the run intersects F. More formally, we have that $T'(S_1, \sigma, S_2)$ holds if and only if $S_2 = \{q_2 \mid (q_1, \sigma, q_2) \in T \text{ and } q_1 \in S_1\}$, and $\mathcal{F} = \{S \mid S \cap F \neq \varnothing\}$.

This idea, which tracks whether there exists a run that ends in an accepting state, cannot be naively applied to Büchi automata, for the obvious reason that accepting runs on infinite words do not end. We use a judicious extension of this idea to determine, soundly and completely, whether there exists an infinite run that visits the set of accepting states infinitely often.

Our key insight is as follows:

Lemma 1.1. *A Büchi automaton $(\Sigma, Q, I, T, \text{BÜCHI}(F))$ accepts $\alpha \in \Sigma^\omega$ if and only if there exist words $u_0, u_1, \ldots \in \Sigma^* \backslash \{\varepsilon\}$, and sets $S_0, S_1, S_2, \ldots \subseteq Q$ such that $S_0 = I$ and:*

1. *The infinite concatenation $u_0 \cdot u_1 \cdot \cdots = \alpha$.*
2. *For all n, and all $q_{n+1} \in S_{n+1}$, there exists $q_n \in S_n$ such that upon reading u_n, q_n has a run to q_{n+1} that visits[2] a state in F.*

Enforcing the second requirement is where the idea of subset tracking plays its part. We prove the lemma before proceeding further.

Proof. **Only If.** If the automaton accepts the word, then we have an accepting run, and we can use the infinitely many indices where it visits accepting sets

[1] This exposition aims to do so in a way that shows the reader (or listener) how they could have come up with the conditions themselves.

[2] By "visits", we mean, "has a transition that enters".

to identify the factors $u_0, u_1, \ldots$, and accordingly choose $S_1, S_2, \ldots$ to be the singleton sets that respectively contain the corresponding state reached in the run.

If. It is slightly more involved to prove that the word has an accepting run if the conditions on $u_0, u_1, \ldots$ and $S_0, S_1, S_2, \ldots$ are met. For this, given $u_0, u_1, \ldots$, and $S_0, S_1, S_2, \ldots$, we construct an infinite graph whose vertices are $\bigcup_{i \in \mathbb{N}, q \in S_i} (q, i)$ i.e., the graph has infinitely many partitions, and the i-th partition has vertices corresponding to the states in S_i. Edges are drawn only between consecutive partitions; there is an edge from (q, i) to $(q', i+1)$ if and only if q, upon reading u_i, has a run to q' that visits a state in F.

Notice that an accepting run corresponds to an infinite path in the graph starting at a vertex of the form $(q, 0)$. To prove that such a path exists, we show that the graph contains an infinite, but finitely branching tree rooted at some vertex of the form $(q, 0)$, and apply König's lemma. [3]

The required tree is easily seen to exist: by construction of the graph, all nodes have finitely many successors. Moreover, by the given conditions, all nodes necessarily have a predecessor. Thus, any node is reachable from some ancestor in the 0-th partition. Since there are only finitely many vertices in the 0-th partition, at least one of them must have infinitely many descendants, giving us an infinite tree.

We can thus conclude that the existence of $u_0, u_1, \ldots$ and $S_0, S_1, S_2, \ldots$ implies the acceptance of the word. $\square$

1.2 Safra Trees: Hierarchical Subset Tracking

As suggested by Lemma 1.1, the idea to follow runs of $(\Sigma, Q, I, T, \text{BÜCHI}(F))$ on $\alpha \in \Sigma^\omega$ is to do subset tracking, but with additional book-keeping to mark *checkpoints* when each state in the current subset could have been reached from some state in the previous checkpoint via an accepting state. Without loss of generality, we shall make the technical assumption that our given Büchi automaton has a unique distinguished initial state with no incoming transitions, i.e., $I = \{q_{\mathsf{init}}\}$.

We explain the core concepts with a simplified illustration.

We start with a single root instance of subset tracking with the beginning of the run as its initial checkpoint, and obtain a sequence of sets $\{q_{\mathsf{init}}\} = S_0, S_1, \ldots$, where for each i, $S_{i+1} = \{q \mid \exists q' \in S_i.\ (q', \alpha(i), q) \in T\}$. A checkpoint will have been reached if for some S_m, all states in S_m can be reached from q_{init} by a run that visits an accepting state.

Our need for additional book-keeping thus arises when S_n intersects F for some n. If $S_n \subseteq F$, then we can immediately mark a checkpoint for our root; however this need not always be the case. Let the intersection be $S'_0 \subset S_n$.

At this point, we create a child instance of subset tracking which begins by recording S'_0. The initial checkpoint for the new tracking is the moment of its

[3] An important and well-known corollary of König's lemma is that every infinite but finitely branching tree has an infinite path.

birth recording S'_0: indeed, by construction, all states in $S'_0 \subseteq F$ can be reached from q_{init} by a run that visits an accepting state.

Now, we run both instances of subset tracking. The subsequent combined trace would be $(S_{n+1}, S'_1), (S_{n+2}, S'_2), \ldots$, where by construction, $S'_j \subseteq S_{n+j}$.

Here, S'_j is the subset of states of S_{n+j} which have runs from q_{init} (the previous checkpoint of the root instance) that visit an accepting state. What happens when $S'_k = S_{n+k}$? This is the moment where all states in the root tracking instance can be reached from an initial state via an accepting state. We hence mark a checkpoint for the root tracking instance. This is also the moment where the book-keeping done by the child instance is rendered redundant. Hence, we delete it.

A few natural questions arise when considering how to generalize this scheme:

- Can the child instance have children of its own? Which tracking instance is forked when S'_j intersects F? The child, the root, or both?
- Is the root tracking instance forked again when S_{n+j} contains an accepting state that is not in S'_j?
- Can *any* tracking instance be adjudged to have reached a checkpoint? If so, when? What should then happen to its descendant instances?

The general construction records the state of our tracking scheme with a (Safra) tree whose nodes correspond to tracking instances, and are moreover ranked by their age. The root instance has rank 1, its oldest child, if it has children, has rank 2, and so on. Indeed, any instance can have children, and be adjudged to have reached a checkpoint: its initial checkpoint is its moment of creation (the root, at creation, only tracks $\{q_{\mathsf{init}}\}$; any other instance, at creation, tracks exclusively accepting states). An instance reaches a checkpoint when every state it tracks has a run from the previous checkpoint via an accepting state: as illustrated above, this is what its children will help it determine.

> Our first goal towards determinization is to construct a tracking scheme such that the word has an accepting run if and only if the scheme, whilst reading the word, creates a tracking instance that celebrates infinitely many checkpoints.

To ensure that there are only finitely many possible trees, and that they suffice for our goal, we enforce some invariants.

For any node v, let S be the subset that it is currently tracking, and let $S^1, \ldots, S^d$ be the subsets its children $v_1, \ldots, v_d$ are tracking. We have that:

- $S^1, \ldots, S^d$ are non-empty and disjoint.
- The union $S' = S^1 \cup \cdots \cup S^d$ is a strict subset of S; if S_0 was the subset that v held at its previous checkpoint, S' is the set of states q such that there is a run that visits an accepting state from some state in S_0 to q.

Analogously to our illustrative example, if we are to maintain the latter invariant, when the subset S held in a node v equals the union S' of the subsets held in its children, the node v celebrates a checkpoint, and we delete all its descendants. We now describe how our tracking scheme works for a given Büchi automaton $(\Sigma, Q, \{q_{\mathsf{init}}\}, T, \textsc{Büchi}(F))$ in more detail.

Recall that the state consists of a tree, whose nodes correspond to subset tracking instances. We define the function $\mathsf{tracking}$ that maps nodes to the subsets they hold. Nodes are ranked by age, and are further annotated with a flag that indicates whether their corresponding tracking instance is celebrating a checkpoint. The initial state is a tree with a single celebrating node root, and $\mathsf{tracking}(\mathsf{root}) = q_{\mathsf{init}}$.

Upon reading a letter σ, the successor state in our tracking scheme is obtained by performing the following steps (we assume that for loops are traversed from oldest to youngest node).

1. Mark each node v as not celebrating.
2. For each node v do $\mathsf{tracking}(v) \leftarrow \{q' \mid q \in \mathsf{tracking}(v) \wedge (q, \sigma, q') \in T\}$.
3. For each node v, create a fresh child v'; $\mathsf{tracking}(v') \leftarrow \mathsf{tracking}(v) \cap F$. Record v' to be the youngest node, and flag it to be celebrating.
4. We now ensure that sibling nodes hold disjoint sets: For each node v, and each node v' that is a younger sibling of v or a descendant of a younger sibling of v, do $\mathsf{tracking}(v') \leftarrow \mathsf{tracking}(v') \setminus \mathsf{tracking}(v)$.
5. For each node v, if $\mathsf{tracking}(v) = \varnothing$, delete v (all its descendants are also guaranteed to be deleted, as they contain subsets of $\mathsf{tracking}(v)$).
6. For each node v, if $\mathsf{tracking}(v)$ is equal to the union of the sets held by its children, then delete all descendants of v, and mark v as celebrating.

The following is seen to hold by construction.

Lemma 1.2. *Consider any node v of the tree that records the state of our tracking scheme, and let $v_1, \ldots, v_d$ be its children (ordered by age, oldest to youngest). We have the following invariants:*

- $\mathsf{tracking}(v_1), \ldots, \mathsf{tracking}(v_d)$ *are non-empty and disjoint.*
- *The union $S' = \mathsf{tracking}(v_1) \cup \cdots \cup \mathsf{tracking}(v_d)$ is a strict subset of $\mathsf{tracking}(v)$; if S_0 was $\mathsf{tracking}(v)$ at its previous checkpoint, S' is the set of states q such that there is a run that visits an accepting state from some state in S_0 to q.*

Definition 1.3. *A node is called persistent if it is never deleted. Observe, in particular, that it is necessarily for a node to be persistent for it to celebrate infinitely often.*

We are now ready to revisit our first goal.

Lemma 1.4. *The automaton $\mathcal{N} = (\Sigma, Q, \{q_{\mathsf{init}}, T, \text{BÜCHI}(F)\})$ accepts $\alpha \in \Sigma^\omega$ if and only if our corresponding tracking scheme, whilst following α, creates a node that celebrates infinitely often.*

Proof. **If.** Assume our tracking scheme, whilst following α, creates a node v that celebrates infinitely often. We shall show that there exist sets $\{q_{\mathsf{init}}\} = S_0, S_1, S_2, \ldots \subseteq Q$ and words $u_0, u_1, \ldots$ (whose infinite concatenation equals α) such that for each i, each state in S_{i+1} has a run on u_i to it from some state in S_i that moreover visits a final state. Applying Lemma 1.1 would then establish the acceptance of α by $\mathcal{N}$.

Indeed, the existence of such sets and words follows directly from the definition of celebrating checkpoints. The sets S_0(iff $v = \mathsf{root}$), $S_1, S_2 \ldots$ are precisely the outputs $\mathsf{tracking}(v)$ at moments where v is celebrating, and $u_0, u_1, \ldots$ are the intervening words. If v is not the root node, we take u_0 be the word that leads to the creation of v, and have that $S_1 \subseteq F$.[4]

Only If. We shall consider an accepting run $\rho = \rho(0)\rho(1)\cdots$ of $\mathcal{N}$ on α, and prove by contradiction that our tracking scheme must create a node that celebrates infinitely often whilst reading α. Observe that the scheme is guaranteed to create *persistent* nodes if there is an accepting run: root is one of them.

Now, suppose towards a contradiction that all persistent nodes celebrate only finitely often. Since only finitely many (in fact at most $|Q|$-many) nodes can co-exist, there exists an index N at which all persistent nodes have been created and have finished celebrating, and all non-persistent nodes that were created before the youngest persistent node have been deleted.

We say that a node v *accounts* for a state q if $q \in \mathsf{tracking}(v)$, it accounts for a run ρ at time j if $\rho(j) \in \mathsf{tracking}(v)$. Given ρ is accepting, we show that if a non-celebrating node v eventually always accounts for ρ, then it has a child v' that eventually always accounts for ρ. Since root always accounts for ρ and assumed to be non-celebrating, and all persistent nodes are also assumed to be non-celebrating, this would imply that there are infinitely many persistent nodes: a contradiction.

Consider a non-celebrating node v that eventually always accounts for ρ (after index N), and consider the first visit ρ makes to an accepting state in this phase. By Lemma 1.2, from this visit on, ρ must always be accounted for by some child of v, because v itself never celebrates. This accountability can only pass from younger to older children, and would eventually need to stay forever with a persistent child v'. $\square$

1.3 Construction of a Deterministic Automaton

We are now equipped to construct a deterministic automaton that has our tracking scheme at its core and recognizes $\mathcal{L}(\mathcal{N})$. The infinite run of a word on this automaton needs to have a mechanism to identify that there is a persistent node

[4]Recall that by convention, a node celebrates upon its creation.

which moreover celebrates infinitely often. The states of our automaton, therefore need to capture information about nodes celebrating, and being deleted.

We observe that in any run of our tracking scheme, from some point on, some i of the oldest nodes will be persistent, and if the j-th oldest node is deleted, then it must be that $j > i$.

This gives us some intuition: in the long run, a node of rank i being deleted is a more decisive event than a node of rank i celebrating; but for $j > i$, a node of rank j being deleted is mitigated by a node of rank i celebrating. We thus assign priorities to events: a node of rank i being deleted is at priority $2i - 1$, this node celebrating is at priority $2i$, e.g. the catastrophe of the root being deleted has priority 1, in case there exists an infinite run, the persistent root celebrating has priority 2, and so on.

> Formulated this way, we ask that the highest priority to be recorded infinitely often be even, i.e., correspond to a celebration.

Suppose the highest priority to be recorded infinitely often is $2i$, for some i. This implies that nodes of rank $1, \ldots, i$ are deleted only finitely often, which means that from some point on, the i oldest nodes are persistent. Among these, the i-th oldest node celebrates infinitely often, implying that $\mathcal{N}$ accepts the word being followed (Lemma 1.4).

Conversely, suppose that the highest priority to be recorded infinitely often is $2i - 1$ for some i. This implies that only $i - 1$ nodes are persistent (if i nodes are persistent, the i-th oldest keeps getting deleted, a contradiction), and none of them celebrate infinitely often, giving us that $\mathcal{N}$ does not accept the word being followed (Lemma 1.4). Similarly, if the highest priority is $2|Q| + 1$, it means that eventually, no node celebrates, and there definitely is no accepting run of $\mathcal{N}$ on the word being followed.

Thus, the state of our automaton will store the state of our tracking scheme (i.e., a Safra tree), and an additional book-keeping component called report, which records the priority of the most important event to have occurred to the nodes of the predecessor tree while obtaining the current tree.

If the report is $2i$, it means that all of the i oldest nodes in the previous tree have been retained in the current tree, and moreover that the i-th oldest node of the previous tree is mapped to a celebrating node in the current tree.

If the report is $2i - 1$, it means that all of the $i - 1$ oldest nodes of the previous tree have been retained, but none of them are mapped to a currently celebrating node, and that the i-th oldest node of the previous tree has been deleted. If the report is $2|Q| + 1$, it means that there have been no celebrations or deletions among the nodes of the predecessor tree.

To accept, we shall require that the highest priority reported infinitely often be even. We now formally record the full construction for completeness' sake.

Construction 1.5 (The Automaton). *Given a complete automaton*

$$\mathcal{N} = (\Sigma, Q, \{q_{\mathsf{init}}\}, T, \text{Büchi}(F)),$$

we construct an equivalent deterministic automaton

$$\mathcal{D} = (\Sigma, \mathsf{Trees}_Q, \{\mathsf{tree}_{\mathsf{init}}\}, \Delta, \textsc{Acc})$$

as follows.

A state $\mathsf{tree} \in \mathsf{Trees}_Q$ *is a tuple*

$$(\mathsf{nodes}, \mathsf{root}, \mathsf{ranking}, \mathsf{parent}, \mathsf{tracking}, \mathsf{report}),$$

where each component is as follows:

- nodes *is a set of (at most $|Q|$) nodes, which form the tree,*
- root *is the distinguished root node of the tree (if non-empty),*
- $\mathsf{ranking} : \mathsf{nodes} \to \{1, \ldots, |Q|\}$ *maps each node to its position when the nodes are sorted by age;* $\mathsf{ranking}(\mathsf{root}) = 1$*,*
- $\mathsf{parent} : \mathsf{nodes}\backslash\{\mathsf{root}\} \to \mathsf{nodes}$ *maps each node to its parent, thus defining the structure of the tree,*
- $\mathsf{tracking} : \mathsf{nodes} \to 2^Q \backslash \{\varnothing\}$*,*
- $\mathsf{report} \in \{1, 2, \ldots, 2|Q| - 1, 2|Q|, 2|Q| + 1\}$*.*

We must have that for all nodes v, v_1, v_2:

- $\left(\bigcup_{u|\mathsf{parent}(u)=v} \mathsf{tracking}(u)\right) \subset \mathsf{tracking}(v)$*,*
- *if* $\mathsf{parent}(v_1) = \mathsf{parent}(v_2)$ *then* $\mathsf{tracking}(v_1) \cap \mathsf{tracking}(v_2) = \varnothing$*.*

The initial state $\mathsf{tree}_{\mathsf{init}}$ *has a single node* root*, such that* $\mathsf{tracking}(\mathsf{root}) = \{q_{\mathsf{init}}\}$*, and* $\mathsf{report} = 2$*.*

The deterministic transition relation Δ is defined as per the program to compute the successor Safra tree. More precisely, given tree_1, σ *the unique tree* tree_2 *such that* $(\mathsf{tree}_1, \sigma, \mathsf{tree}_2) \in \Delta$ *is obtained by following steps (for loops assumed to operate from oldest to youngest node).*

1. $\mathsf{tree}_2 \leftarrow \mathsf{tree}_1$, $\mathsf{rank}_{2,temp} \leftarrow \mathsf{rank}_1$.
2. *For all* $v \in \mathsf{nodes}_2$, $\mathsf{tracking}_2(v) \leftarrow \{q' \mid q \in \mathsf{tracking}_2(v) \wedge (q, \sigma, q') \in T\}$.
3. *For all* $v \in \mathsf{nodes}_2$*, add v' to* nodes_2*;* $\mathsf{tracking}_2(v) \leftarrow \mathsf{tracking}_2(v) \cap F$, $\mathsf{rank}_{2,temp}(v') \leftarrow |\mathsf{nodes}_2| + 1$, $\mathsf{parent}_2(v') \leftarrow v$.
4. *For all* $v, v' \in \mathsf{nodes}_2$*, if v' is a younger sibling of v, or a descendant of a younger sibling of v,* $\mathsf{tracking}_2(v') \leftarrow \mathsf{tracking}_2(v') \setminus \mathsf{tracking}_2(v)$.
5. *Let* $\mathsf{HighestCelebrate}$ *be the minimum i such that* $\mathsf{rank}_{2,temp}(v) = i$ *and* $\mathsf{tracking}_2(v) = \bigcup_{u|\mathsf{parent}_2(u)=v} \mathsf{tracking}_2(u)$.

6. *Let* $\mathsf{HighestDelete}$ *be the minimum* i *such that* $\mathsf{rank}_{2,temp}(v) = i$ *and either* $\mathsf{tracking}_2(v) = \varnothing$*, or some ancestor* v' *satisfies* $\mathsf{tracking}_2(v') = \bigcup_{u \mid \mathsf{parent}_2(u)=v'} \mathsf{tracking}_2(u)$.

7. *For all* $v \in \mathsf{nodes}_2$ *such that* $\mathsf{tracking}_2(v) = \bigcup_{u \mid \mathsf{parent}_2(u)=v} \mathsf{tracking}_2(u)$*, delete all descendants of* v.

8. *For all* $v \in \mathsf{nodes}_2$ *such that* $\mathsf{tracking}_2(v) = \varnothing$*, delete* v.

9. *Define* rank_2 *to be the order of the remaining* nodes_2*, when sorted by* $\mathsf{rank}_{2,temp}$.

10. $\mathsf{report}_2 \leftarrow \min(2|Q|+1, 2 \cdot \mathsf{HighestCelebrate}, 2 \cdot \mathsf{HighestDelete} - 1)$.

Finally, we define

$$\textsc{Acc} = \{\tau \mid \tau \in \mathsf{Trees}_Q^\omega \wedge \liminf_n (\tau(n).\mathsf{report}) \text{ is even}\},$$

i.e., the smallest number (highest priority) reported infinitely often is even.

Through our construction, we have discovered what is called a deterministic *parity* automaton over infinite words.

Definition 1.6 (Parity Automata)**.** *A parity automaton is given as*

$$(\Sigma, Q, I, T, \textsc{parity}(\mathcal{F})),$$

where the parity acceptance condition is given by a partition $\mathcal{F}$ *of* Q *into sets* $F_1, F_2, \ldots, F_d$*, and is defined to be satisfied by a run* $\rho \in Q^\omega$ *if*[5]

$$\min\{i \mid \mathit{Inf}(\rho) \cap F_i \neq \varnothing\} \text{ is even.}$$

We have proven the following through our construction and its correctness.

Theorem 1.7. *For every Büchi automaton* $\mathcal{N}$*, we can construct a deterministic parity automaton* $\mathcal{D}$ *such that* $\mathcal{L}(\mathcal{N}) = \mathcal{L}(\mathcal{D})$.

As we shall discuss later in the course, the corresponding parity *games* form perhaps the most interesting class of infinite games, and play a key role in verification and synthesis. Finding fast algorithms for parity games is an active research topic, with several (albeit as yet impractical) quasi-polynomial-time algorithms.

[5]Some sources use the maximum instead. The two conventions are obviously and easily interchangeable, and their relative convenience depends on the context. We will clarify if we use the alternate convention in another context.

1.4 Complementation

To prove that Büchi automata, and hence ω-regular languages are closed under complementation, we shall establish two facts: (1) deterministic parity automata are closed under complementation; (2) for every deterministic parity automaton, we can construct an equivalent Büchi automaton. The latter would also prove that deterministic parity automata recognize precisely the class ω-regular languages.

The first fact is quite easily observed.

Construction 1.8. *The complement of a complete and deterministic parity automaton*

$$\mathcal{D} = (\Sigma, Q, \{q_{\mathsf{init}}\}, T, \textsc{parity}(F_1, \dots, F_d))$$

is

$$\mathcal{D}' = (\Sigma, Q, \{q_{\mathsf{init}}\}, T, \textsc{parity}(F'_1, \dots, F'_d, F'_{d+1})),$$

where $F'_1 = \varnothing$, *and for* $i > 1$, $F'_i = F_{i-1}$.

The idea to construct a Büchi automaton $\mathcal{N}$ that is equivalent to a given deterministic parity automaton $\mathcal{D}$ is for $\mathcal{N}$ is to initially emulate $\mathcal{D}$, nondeterministically guess an even index i, and subsequently run on an incomplete, but deterministic partial emulation of $\mathcal{D}$ with states $F_i \cup \dots \cup F_d$. The guess is validated if the subsequent run is infinite, and visits F_i infinitely often.

Construction 1.9. *Given a parity automaton*

$$\mathcal{A} = (\Sigma, Q, \{q_{\mathsf{init}}\}, \Delta, \textsc{parity}(F_1, \dots, F_d)),$$

we can construct an equivalent Büchi automaton

$$\mathcal{N} = (\Sigma, Q', \{q_{\mathsf{init}}\}, T, \textsc{Büchi}(F')),$$

where

$$Q' = Q \cup \bigcup_{1 \le i \le \lfloor d/2 \rfloor} (F_{2i} \cup F_{2i+1} \dots \cup F_d) \times \{2i\},$$

$$\mathsf{guess} = \{(q, \sigma, (q', 2i)) \mid (q, \sigma, q') \in \Delta \wedge q' \in F_j \textit{ where } j \ge 2i\},$$

$$\mathsf{validate} = \{((q, 2i), \sigma, (q', 2i)) \mid (q, \sigma, q') \in \Delta \wedge q \in F_j \wedge q' \in F_{j'} \textit{ where } j, j' \ge 2i\},$$

$$T = \Delta \cup \mathsf{guess} \cup \mathsf{validate},$$

$$F' = \{(q, 2i) \mid q \in F_{2i}\}.$$

We have thus established:

Theorem 1.10. *Deterministic parity automata recognize precisely the class of ω-regular languages.*

Theorem 1.11. *The class of ω-regular languages is closed under complementation.*

Theorem 1.12. *Given a Büchi automaton $\mathcal{N}$, we can construct a Büchi automaton $\mathcal{N}'$ such that $\mathcal{L}(\mathcal{N}')$ is the complement of $\mathcal{L}(\mathcal{N})$.*

Proof. We use Construction 1.5 to get a deterministic parity automaton $\mathcal{D}$ that is equivalent to $\mathcal{N}$; Construction 1.8 to then complement $\mathcal{D}$ to $\mathcal{D}'$; and finally Construction 1.9 to obtain a Büchi automaton $\mathcal{N}'$ equivalent to $\mathcal{D}'$. $\square$

1.5 Deterministic Automata: Muller Acceptance

One of the pieces of intuition we gained through Construction 1.5 is that if a class of deterministic automata is to be as powerful as Büchi automata, then the acceptance condition must express not only that desirable states are visited infinitely often, but also that undesirable states are visited only finitely often.

It is sometimes easy to work with the verbose Muller acceptance condition, which enforces such requirements explicitly, as opposed to the implicit enforcement by the parity acceptance condition.

Definition 1.13 (Muller Automata)**.** *A Muller automaton is given as*

$$(\Sigma, Q, I, T, \textsc{Muller}(\mathcal{F})),$$

where the Muller acceptance condition is given by a table $\mathcal{F} = \{F_1, \ldots, F_d\}$, where $F_1, \ldots, F_d \subseteq Q$, and is defined to be satisfied by a run $\rho \in Q^\omega$ if

$$\mathit{Inf}(\rho) \in \mathcal{F}.$$

As an example, consider the deterministic Muller automaton over the alphabet $\{a, b\}$ with states q_a, q_b, initial state q_a, for all q, $(q, a, q_a) \in T$ and $(q, b, q_b) \in T$. If the acceptance condition is $\textsc{Muller}(\{\{q_a\}, \{q_a, q_b\}\})$, it recognizes the language of words with infinitely many a's. If the acceptance condition is $\textsc{Muller}(\{\{q_b\}\})$ instead, it recognizes the language of words with finitely many a's.

Exercise 1.14. *Given and complete and deterministic Muller automaton recognizing the language $L \subseteq \Sigma^\omega$, show how to construct a complete and deterministic Muller automaton recognizing the language $\Sigma^\omega \backslash L$.*

Exercise 1.15. *Given a complete and deterministic parity automaton $\mathcal{D}_P$, show how to construct an equivalent deterministic Muller automaton $\mathcal{D}_M$.*

Exercise 1.16. *Given a Muller automaton $\mathcal{A}$, show how to construct an equivalent Büchi automaton $\mathcal{N}$.*

Hint: As in Construction 1.9, $\mathcal{N}$ needs to guess the set of infinitely visited states, and validate its guess. However, during the validation, it needs to ensure that not only are states outside the guessed set inaccessible, but also that states within the guessed set are visited infinitely often. For this, consider using the book-keeping trick from the construction to intersect Büchi automata.

We conclude this section with a language-theoretic fact, which is most readily proven through Muller automata.

Theorem 1.17. *A language $L \subseteq \Sigma^\omega$ is ω-regular if and only if it a Boolean combination of languages $\overrightarrow{W_1}, \dots, \overrightarrow{W_d}$ where $W_1, \dots, W_d \subseteq \Sigma^*$ are regular.*

Proof. **If.** The languages $\overrightarrow{W_1}, \dots, \overrightarrow{W_d}$ are recognizable by deterministic Büchi automata, and hence clearly ω-regular. We have shown that the class of ω-regular languages is closed under Boolean operations, and hence deduce that L must be ω-regular.

Only If. We take as our premise that L is an ω-regular language, and obtain a complete and deterministic Muller automaton $(\Sigma, Q, \{q_{\mathsf{init}}\}, \Delta, \textsc{Muller}(\mathcal{F}))$ that recognizes it. Now, for $q \in Q$, we define W_q to be the regular language accepted by the deterministic finite-word automaton $(\Sigma, Q, \{q_{\mathsf{init}}\}, \Delta, \{q\})$, and observe that the run of the Muller automaton on a word α visits a state q infinitely often if and only if $\alpha \in \overrightarrow{W_q}$. By the definition of the Muller acceptance condition, we have that

$$L = \bigcup_{F \in \mathcal{F}} \left(\left(\bigcap_{q \in F} \overrightarrow{W_q} \right) \cap \left(\bigcap_{q \notin F} (\Sigma^\omega \setminus \overrightarrow{W_q}) \right) \right),$$

which is a Boolean combination of limits of regular languages, as desired. $\square$