5.12 Finite Automata, Infinite Strings: Büchi Automata

So far, we have considered, as input to our machines, only strings of finite length. Thus we have focused on problems for which we expect to write programs that read an input, compute a result, and halt. Many problems are of that sort, but some are not. For example, consider:

- An operating system.
- An air traffic control system.
- A factory process control system.

Ideally, such systems never halt. They should accept an infinite string of inputs and continue to function. Define $\Sigma^\infty$ to be the set of infinite length strings drawn from the alphabet $\Sigma$. For the rest of this discussion, define a language to be a set of such infinite-length strings.

To model the behavior of processes that do not halt, we can extend our notion of an NDFS to define a machine whose inputs are elements of $\Sigma^\infty$. Such machines are sometimes called $\omega$-automata (or omega automata).

We'll define one particular kind of $\omega$-automaton: A Büchi automaton is a quintuple $(K, \Sigma, \Delta, S, A)$, where:

- $K$ is a finite set of states.
- $\Sigma$ is the input alphabet.
- $S \subseteq K$ is a set of start states.
- $A \subseteq K$ is the set of accepting states.
- $\Delta$ is the transition relation. It is a finite subset of:

$$(K \times \Sigma) \times K.$$

Note that, unlike NDFSMs, Büchi automata may have more than one start state. Note also that the definition of a Büchi automaton does not allow $\epsilon$-transitions.

We define configuration, initial configuration, yields-in-one-step, and yields exactly as we did for NDFSMs. A computation of a Büchi automaton $M$ is an infinite sequence of configurations $C_0, C_1, \ldots$ such that:

- $C_0$ is an initial configuration, and
- $C_0 \not\rightarrow_M C_1 \not\rightarrow_M C_2 \not\rightarrow_M \ldots$
But now we must define what it means for a Büchi automaton $M$ to accept a string. We can no longer define acceptance by the state of $M$ when it runs out of input, since it won’t. Instead, we’ll say that $M$ accepts a string $w$ if, in at least one of its computations, there is some accepting state $q$ such that, when processing $w$, $M$ enters $q$ an infinite number of times. So note that it is not required that $M$ enter an accepting state and stay there. But it is not sufficient for $M$ to enter an accepting state just once (or any finite number of times). As before, the language accepted by $M$, denoted $L(M)$, is the set of all strings accepted by $M$. A language $L$ is Büchi-acceptable iff it is accepted by some Büchi automaton.

Büchi automata can be used to model concurrent systems, hardware devices, and their specifications. These programs called model checkers can verify that those systems correctly conform to a set of stated requirements. (H1.2)

**EXAMPLE 5.38 Büchi Automata for Event Sequences**

Suppose that there are five kinds of events that can occur in the system that we wish to model. We’ll call them $a$, $b$, $c$, $d$, and $e$. So let $S = \{a, b, c, d, e\}$.

We first consider the case in which we require that some set of events occur. The following (nondeterministic) Büchi automaton accepts all and only the elements of $2^S$ that contain at least one occurrence of $e$.

```
    c  d  e
    1   2
```

Next suppose that we require that there come a point after which only a subset occurs. The following Büchi automaton (designed using our assumption that the dead state need not be written explicitly) accepts all and only the elements of $2^S$ that eventually reach a point after which no events other than $e$ occur.

```
    c  d  e
    1   2
```

Finally, suppose that we require that every event be immediately followed by an $e$ event. The following Büchi automaton (this time with the dead state, shown explicitly) accepts all and only the elements of $2^S$ that satisfy that requirement.

```
    c  d  e
    1   2
```

**EXAMPLE 5.39 Mutual Exclusion**

Suppose that we want to model a concurrent system with two processes and enforce the constraint often called mutual exclusion: namely, that no two processes are in their critical region at the same time. We could do this in the usual way using an abstract machine approach (see Example 5.38), where the system requires the input from at least two intervals in which both processes are in their critical region but the input from at least one other interval. But a more direct way to model this situation is to use Büchi automata. Consider the automaton below which models a system with two processes. One process will be in its critical region if the other process is in its critical region. The system is constructed from two Büchi automata connected by an accept state transition $T(\alpha_1, \alpha_2)$. (The transition is enabled when both processes are in their critical region, which will be modeled by the Büchi automaton.

```
    0
    1
    (0, 1)
```

While there is an obvious similarity between Büchi automata and FSMs, and the languages they accept are related, as described below, there is one important difference. For Büchi automata, non-determinism matters.
EXAMPLE 5.40 For Büchi Automata, Nondeterminism Matters

Let \( L = \{ w \in \{ a, b \}^* : t_w(w) \text{ is finite} \} \). Note that every string in \( L \) must contain an infinite number of \( a \)'s. The following nondeterministic Büchi automaton accepts \( L \):

![Diagram of a Büchi automaton accepting \( L \)]

We can try to build a corresponding deterministic machine by using the construction that we used in the proof of Theorem 5.3 (which says that for every NDFSM there does exist an equivalent DFSM). The states of the new machine will then correspond to subsets of states of the original machine and we’ll have:

![Diagram of a deterministic Büchi automaton]

This new machine is indeed nondeterministic and it does accept all strings in \( L \). Unfortunately, it also accepts in infinite number of strings that are not in \( L \), including \( (aa)^n \). More unfortunately, we cannot do any better.

THEOREM 5.7 Nondeterministic versus Deterministic Büchi Automata

**Theorem:** There exist languages that can be accepted by a nondeterministic Büchi automaton (i.e., one that meets the definition we have given), but for which there exists no equivalent deterministic Büchi automaton (i.e., one that has a single start state and whose transitions are defined by a function from \( \mathbb{K} \times \Sigma \) to \( \mathbb{K} \)).

**Proof:** The proof is by a demonstration that no deterministic Büchi automaton accepts the language \( L = \{ w \in \{ a, b \}^* : t_w(w) \text{ is finite} \} \) of Example 5.40. Suppose that there were such a machine \( B \). Then, among the strings accepted by \( B \), would be every string of the form \( w(a^n) \), where \( w \) is some finite string in \( \{ a, b \}^* \). This must be true since all such strings contain only a finite number of \( b \)'s. Remove from \( B \) any states that are not reachable from the start state. Now consider any remaining state \( q \) in \( B \). Since \( q \) is reachable from the start state, there must exist at least one finite string that drives \( B \) from the start state to \( q \). Call that string \( w \). Then, as we just observed, \( w(a^n) \) is in \( L \) and so must be accepted by \( B \). In order for \( B \) to accept it, there must be at least one accepting state \( q_a \) that occurs infinitely often in the computation of \( B \) on \( w(a^n) \). That accepting state must be reachable from \( q \) (the state of \( B \) when just \( w \) has been read) by some finite number, which we’ll call \( q \), of \( a \)'s (since \( B \) has only a finite number of states). Compute \( q \) for every state \( q \) in \( B \). Let \( m \) be the maximum of the \( q \) values.

We can now show that \( B \) accepts the string \( (ba^n)^* \), which is not in \( L \). Since \( B \) is deterministic, its transition function is defined on all (state, input) pairs, so it must run forever on all strings including \( (ba^n)^* \). From the last paragraph we know that, from any state, there is a string of \( m \) or fewer \( a \)'s that can drive \( B \) to an accepting state. So, in particular, after each time it reads a \( b \), followed by a sequence of \( a \)'s, \( B \) must reach some accepting state within \( m \) a's. But \( B \) has only a finite number of accepting states. So, on input \( (ba^n)^* \), \( B \) reaches some accepting state an infinite number of times and it accepts.

There is a natural relationship between the languages of infinite strings accepted by Büchi automata and the regular languages (i.e., the languages of finite strings accepted by FSMs). To describe this relationship requires an understanding of the closure properties of the regular languages that we will present in Section 8.3, as well as some of the decision procedures for regular languages that we will present in Chapter 9. It would be helpful to read those sections before continuing to read this discussion of Büchi automata.

Any Büchi-acceptable language can be described in terms of regular languages. To see how, observe that any Büchi automaton \( B \) can be almost be viewed as an FSM, if we simply consider input strings of finite length. The only reason that can’t quite be done is that Büchi automata may have multiple start states. So, from any Büchi automaton \( B \), we can build what we call the mirror FSM \( M \) to \( B \) as follows: Let \( M = B \) except that, if \( B \) has more than one start state, then, in \( M \), create a new start state that has an \( a \)-transition to each of the start states of \( B \). Notice that the set of finite length strings that drives \( B \) from a start state to some state \( q \) is identical to the set of finite length strings that can drive \( M \) from its start state to state \( q \).

Now consider any Büchi automaton \( B \) and any string \( w \) that \( B \) accepts. Since \( w \) is accepted, there is some accepting state \( q \) in \( B \) that it visited an infinite number of times while \( B \) processes \( w \). Call that state \( q \). (There may be more than one such state. Pick one.) Then we can divide \( w \) into two parts, \( x \) and \( y \). The first part, \( x \), has finite length and it drives \( B \) from a start state to \( q \) for the first time. The second part, \( y \), has infinite length and it simply pushes \( B \) through one loop after another, each of which stays in \( q \) (although there may be more than one path that does this). The set of possible values for \( x \) is regular: It is exactly the set that can be accepted by the FSM \( M \) that mirrors \( B \), if we let \( q \) be \( M \)'s only accepting state. Call a path from \( q \) back to itself minimal if it does not pass through \( q \). Then we also notice that the set of strings that can force \( B \) through such a minimal path is also regular. It is the set accepted by the FSM \( M \) that mirrors \( B \), if we let \( q \) be both \( M \)'s start state and its only accepting state. These observations lead to the following theorem:
THEOREM 5.8 Büchi-Acceptable and Regular Languages

Theorem: If \( L \) is a Büchi-acceptable language, then \( L \) is the finite union of sets each of which is of the form \( XY^* \), where each \( X \) and \( Y \) is a regular language.

Proof: Given any Büchi automaton \( B = (K, \Sigma, \delta, S, A) \), let \( W_{eq} \) be the set of all strings that drive \( B \) from state \( q_0 \) to state \( q \). Then, by the definition of what it means for a Büchi automaton to accept a string, we have:

\[
L(B) = \bigcup_{q \in S} W_{eq} Y^*.
\]

If \( L \) is a Büchi-acceptable language, then there is some Büchi automaton \( B \) that accepts it. So the only-if part of the claim is true since:

- \( S \) and \( A \) are both finite,
- For each \( s \) and \( q \), \( W_{eq} \) is regular since it is the set of strings accepted by \( B \)'s mirror FinSM \( M \) with start state \( s \) and single accepting state \( q \),
- \( W_{eq} = Y^* \), where \( Y \) is the set of strings that can force \( B \) along a minimal path from \( q \) back to \( q \),
- \( Y \) is regular since it is the set of strings accepted by \( B \)'s mirror FinSM \( M \) with \( q \) as its start state and its only accepting state, and
- The regular languages are closed under Kleene star so \( W_{eq} = Y^* \) is also regular.

The if part follows from a set of properties of the Büchi-acceptable and regular languages that are described in Theorem 5.9.

THEOREM 5.9 Closure Properties of Büchi Automata

Theorem and Proof: The Büchi-acceptable languages (like the regular languages) are closed under:

- Concatenation with a regular language: If \( L_1 \) is a regular language and \( L_2 \) is a Büchi-acceptable language, then \( L_1 L_2 \) is Büchi-acceptable. The proof is similar to the proof that the regular languages are closed under concatenation except that, since \( s \) transitions are not allowed, the machines for the two languages must be "glued together" differently. If \( q \) is a state in the FinSM that accepts \( L_1 \), and there is a transition from \( q \), labeled \( c \), to some accepting state, then add a transition from \( q \), labeled \( c \), to each start state of the Büchi automaton that accepts \( L_2 \).
- Union: If \( L_1 \) and \( L_2 \) are Büchi-acceptable, then \( L_1 \cup L_2 \) is also Büchi-acceptable. The proof is analogous to the proof that the regular languages are closed under union. Again, since \( s \) transitions are not allowed, we must use a slightly different glue. The new machine we will build will have transitions directly from a new start state to the states that the original machines can reach after reading one input character.
- Intersection: If \( L_1 \) and \( L_2 \) are Büchi-acceptable, then \( L_1 \cap L_2 \) is also Büchi-acceptable. The proof is by construction of a Büchi automaton that effectively runs a Büchi automaton for \( L_1 \) in parallel with one for \( L_2 \).
- Complement: If \( L \) is Büchi-acceptable, then \( \bar{L} \) is also Büchi-acceptable. The proof of this claim is less obvious. It is given in [Thomas 1990].

Büchi automata are useful as models for computer systems whose properties we wish to reason about because some interesting questions can be answered about them. In particular, Büchi automata share with FSMS the existence of decision procedures for all of the properties described in the following theorem:

THEOREM 5.10 Decision Procedures for Büchi Automata

Theorem: There exist decision procedures for all of the following properties:

- Emptiness: Given a Büchi automaton \( B \), is \( L(B) \) empty?
- Nondeterminacy: Given a Büchi automaton \( B \), is \( L(B) \) non-empty?
- Inclusion: Given two Büchi automata \( B_1 \) and \( B_2 \), is \( L(B_1) \subseteq L(B_2) \)?
- Equivalence: Given two Büchi automata \( B_1 \) and \( B_2 \), is \( L(B_1) = L(B_2) \)?

Proof: The proof of each of these claims can be found in [Thomas 1990].

Exercises

1. Give a clear English description of the language accepted by the following DFSM: