Chapter 11

Decidability for Regular Languages

11.1 Introduction

We have three basic questions to answer:

1. How can we tell if two regular expressions define the same language?
2. How can we tell if two FA's are equivalent?
3. How can we tell if the language defined by an FA has finitely many or infinitely many words in it?

Note that questions 1 and 2 are essentially the same by Kleene's Theorem.

11.2 Decidable Problems

Definition: A problem is *effectively solvable* if there is an algorithm that provides the answer in a finite number of steps, no matter what the particular inputs are (but may depend on the size of the problem).

The maximum number of steps the algorithm will take must be predictable before we begin executing the procedure.

Example: Problem: find roots of quadratic equation $ax^2 + bx + c = 0$.
Solution: use quadratic equation

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

No matter what the coefficients a, b, and c are, we can compute the solution using the following operations:

- four multiplications
- two subtractions
- one square root
- one division

Another solution: keep guessing until we find a root.
This approach is not guaranteed to find root in a fixed number of steps.

Example: Find the maximum of n numbers. An effective solution for this is to scan through the list once while updating the maximum observed thus far. This takes $O(n)$ steps.

Definition: An effective solution to a problem that has a yes or no answer is called a *decision procedure*. A problem that has a decision procedure is called *decidable*.

11.2.1 Is $L_1 = L_2$?

Determine if two languages L_1 and L_2 are the same:

- Method 1: Check if the language

  $$L_3 \equiv (L_1 \cap L_2') + (L_1' \cap L_2)$$

  has any words (even Λ).
- If $L_1 = L_2$, then $L_3 = \emptyset$.

- If $L_1 \neq L_2$, then $L_3 \neq \emptyset$.

  Example: Suppose $L_1 = \{a, aa\}$ and $L_2 = \{a, aa, aaa\}$. Then $L_1 \cap L_2' = \emptyset$, but $L_1' \cap L_2 = \{aaa\}$. Thus, $L_1 \neq L_2$.

- So now we have reduced the problem of determining if $L_1 = L_2$ to determining if $L_3 = \emptyset$.

11.2.2 Is $L = \emptyset$?

- So we need a method for determining if a regular language is empty.
- Since the language is regular, it has a regular expression and a FA.
- Given a regular expression, check if there is any part that is not concatenated with $\emptyset$.
- Specifically, use the following algorithm to determine if $L = \emptyset$ given a regular expression r for L:
- **Method 1** (for deciding if a language $L = \emptyset$ given regular expression r for L):
  - Write r as
    $$r = r_1 + r_2 + \cdots + r_n,$$
    where for each $i = 1, 2, \ldots, n$, $r_i = r_{i,1} r_{i,2} \cdots r_{i,j_i}$ for some $j_i \geq 1$; i.e., r is written as a "sum" of other regular expressions r_i, $i = 1, 2, \ldots, n$, where each r_i is a concatenation of regular expressions. It is always possible to write any regular expression r in this form.
  - If there exists some $i = 1, 2, \ldots, n$ such that $r_{i,j} \neq \emptyset$ for all $1 \leq j \leq j_i$, then $L \neq \emptyset$. In other words, if one of the summands has none of its "factors" being $\emptyset$, then the language L is not empty.
  - If for each $i = 1, 2, \ldots, n$, at least one of $r_{i,1}, r_{i,2}, \ldots, r_{i,j_i}$ is $\emptyset$, then $L = \emptyset$. In other words, if each of the summands has at least one "factor" being $\emptyset$, then the language L is empty.

  Example: The regular expression
  $$\emptyset(\mathbf{b} + \mathbf{a})^* + \mathbf{b}$$

has the last **b** not concatenated with $\emptyset$ so the language is not empty.

Example: The regular expression

$$\emptyset(\mathbf{b}+\mathbf{a})^* + \emptyset\mathbf{b}$$

has all parts concatenated with $\emptyset$ so the language is empty.

Remarks: The algorithm in the book for determining if $L = \emptyset$ given a regular expression for L is incorrect.

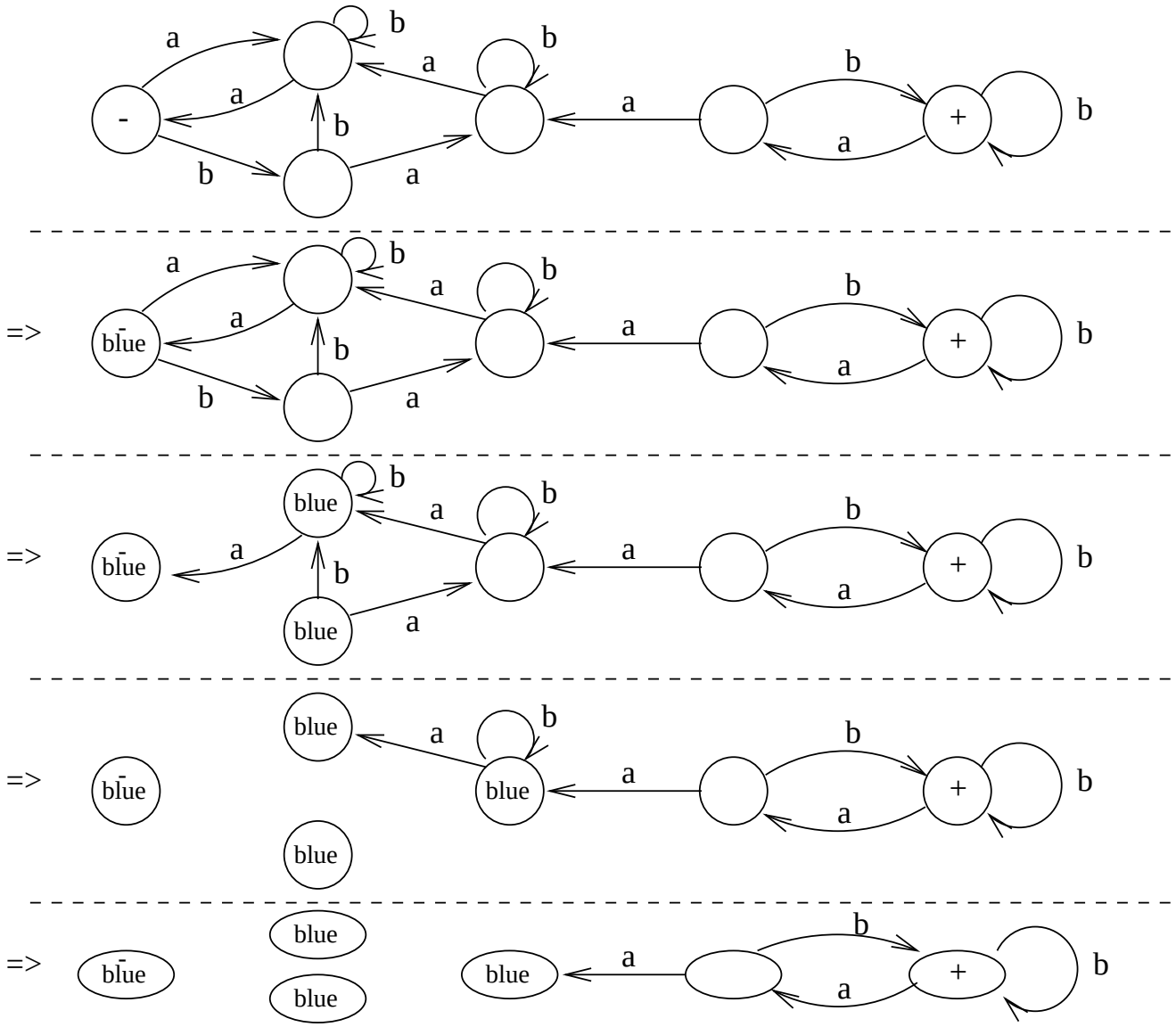
- **Method 2** (for deciding if a language $L = \emptyset$): Given an FA, we check if there are any paths from $-$ to some $+$ state by using the "blue paint algorithm":

  1. Paint the start state blue.
  2. From every blue state, follow each edge that leads out of it and paint the connecting state blue, then delete this edge from the machine.
  3. Repeat Step 2 until no new state is painted blue, then stop.
  4. When the procedure has stopped, if any of the final states are painted blue, then the machine accepts some words, and if not, the machine accepts no words.

  Remarks on Method 2:

  - The above algorithm will iterate Step 2 at most N times, where N is the number of states in the machine.
  - Thus, it is a decision procedure.

Example:



Theorem 17 *Let F be an FA with N states. Then if F accepts any strings at all, it accepts some string with $N-1$ or fewer letters.*

Proof.

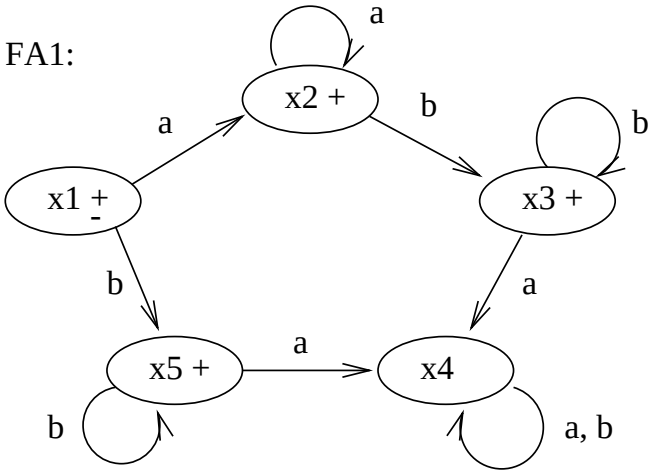
- Consider any string w that is accepted by F.
- Let $s = w$ and DONE = NO.
- Do while (DONE == NO)
  - Trace path of s through F.
  - If no circuits in path, then set DONE = YES.
  - If there are circuits in the path, then
    - Eliminate first circuit in the path.
    - Let s be the string resulting from the new path.
- Resulting path:
  - Starts in initial state.
  - Ends in a final state.
  - Has no circuits, so visits at most N states.
  - This corresponds to a string of at most $N-1$ letters.
  - String is accepted by FA.

■

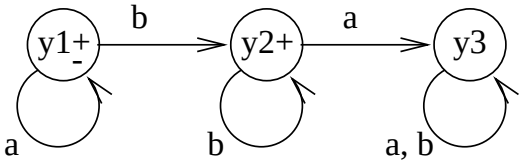
- **Method 3** (for deciding if a language $L = \emptyset$): Test all words with $N-1$ or fewer letters by running them on the FA.

Example: Consider the languages L_1 and L_2 with FA's:

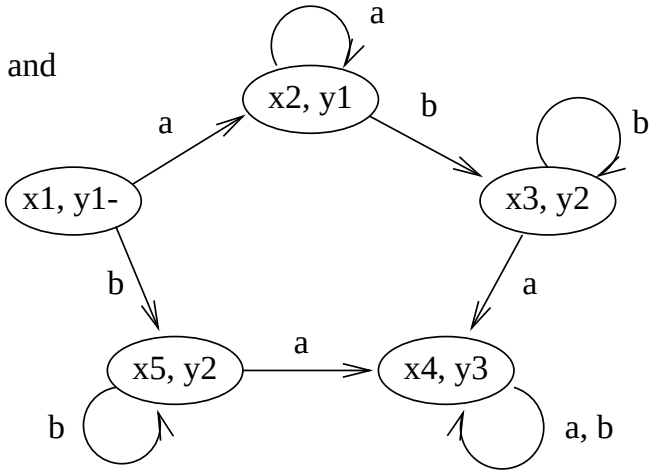
FA1:



FA2:



FA for both
L1 ∩ L2' and
L1' ∩ L2



Theorem 18 *There are effective procedures to decide whether:*

1. *A given FA accepts any words.*
2. *Two FA's are equivalent; i.e., the two FA's accept the same language.*
3. *Two regular expressions are equivalent; i.e., the two regular expressions generate the same language.*

Remarks:

- We can establish part 3 of Theorem 18 by first converting the regular expressions into FA's.
- We previously saw an effective procedure for doing this in the proof of Kleene's Theorem.
- Then we just developed an effective procedure to decide whether two FA's are equivalent.

11.2.3 Is L infinite?

Determining if a language L is infinite

- If we have a regular expression for L, then all we need to do is check if the $*$ is applied to some part of the regular expression that is not Λ nor $\emptyset$.
- Note that $\Lambda^* = \Lambda$ and $\emptyset^* = \Lambda$.
- Note that $\mathbf{a}^*$ is infinite.

Theorem 19 *Let F be an FA with N states. Then*

1. *If F accepts an input string w such that*
$$N \leq length(w) < 2N$$
then F accepts an infinite language.
2. *If F accepts infinitely many words, then F accepts some word w such that*
$$N \leq length(w) < 2N$$

Proof.

1.
   - Assume that F accepts an input string w such that
     $$N \leq \text{length}(w) < 2N$$
   - Since $\text{length}(w) \geq N$, the second version of the pumping lemma (Theorem 14) implies that there exist substrings x, y, and z such that $y \neq \Lambda$ and xy^nz, $n = 0, 1, 2, \ldots$, are all accepted by F.
   - Thus, the FA accepts infinitely many words.

2.
   - Assume that F accepts infinitely many words.
   - This implies that there exists some word u accepted by F that has a circuit (possibly more than one). Why?
   - Each circuit can consist of at most N states since F has only N states.
   - Iteratively eliminate the first circuit in the path until only one circuit left (as in the proof of Theorem 17).
   - Let v correspond to the word from this one-circuit path, and note that v is accepted by F.
   - We can write v as the concatenation of three strings x, y, and z, i.e.,
     $$v = xyz,$$
     such that
     - x consists of the letters read before the circuit.
     - y consists of the letters read along the circuit.
     - z consists of the letters read after the circuit
   - We can show that
     $$0 < \text{length}(y) \leq N$$
     as follows:
     - Since we have eliminated all but the first circuit, the circuit starts and ends in the same state and all of the other states are unique.
     - Thus, the circuit can visit at most $N + 1$ states (with at most one state repeated).
     - This corresponds to reading at most N letters.

  - Also, since a circuit corresponds to at least one transition and each transition in an FA uses up exactly one letter, we see that $\text{length}(y) > 0$.

- We can show that

$$\text{length}(x) + \text{length}(z) < N$$

  as follows:

  - Since we constructed the string v by eliminating all but the first circuit, the paths followed by processing x and z have no circuits.
  - Thus, all of the states visited along the paths followed by processing x and z are unique.
  - Hence, the paths followed by processing x and z visit at most N states.
  - This means that $\text{length}(x) + \text{length}(z) \leq N - 1 < N$.

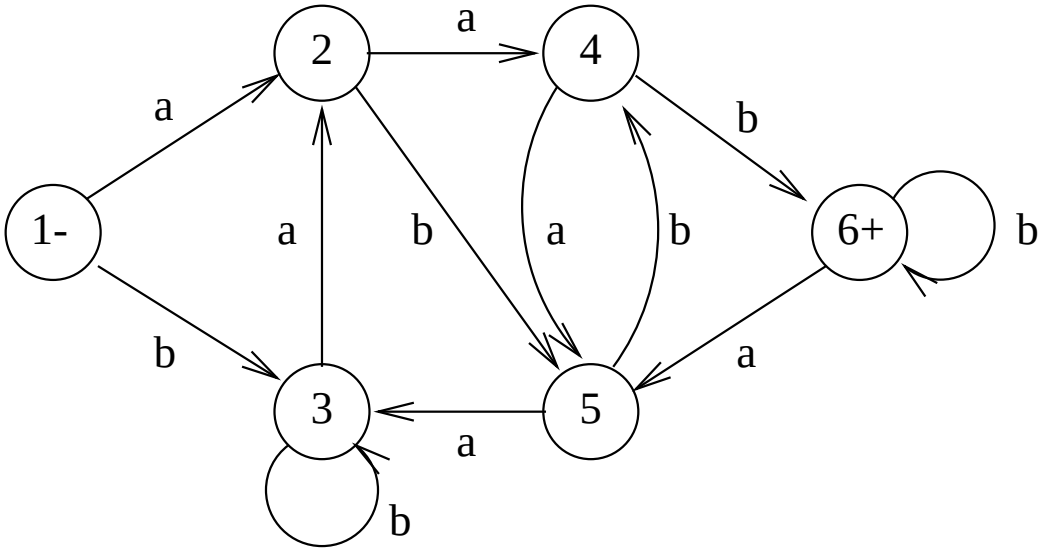
- Thus,

$$\text{length}(v) = \text{length}(x) + \text{length}(y) + \text{length}(z) \leq N - 1 + N < 2N.$$

- If v has at least N letters, then we are done.
- If v has less than N letters, then we can pump up the cycle some number of times to obtain a word that has the desired characteristics since $0 < \text{length}(y) \leq N$.

■

Example:



Consider the word $w = abaaaababbabb$

- length$(w) = 13 > 2N = 12$.
- w is accepted by the FA.
- Processing w on FA takes the path

$$1 \to \underbrace{2 \to 5 \to 3}_{\text{circuit 1}} \to 2 \to \underbrace{4 \to 5}_{\text{circuit 2}} \to \underbrace{4 \to 5}_{\text{circuit 3}} \to \underbrace{4 \to 6 \to 5}_{\text{circuit 4}} \to 4 \to 6$$

- Bypassing all but the first circuit yields the path

$$1 \to 2 \to 5 \to 3 \to 2 \to 4 \to 6$$

  which corresponds to the word $abaaab$, which has length 6.

- Thus, Theorem 19 implies that the FA accepts an infinite language.

Consider the word $w = bbaabb$

- length$(w) = 6 = N$
- w is accepted by the FA.

- Processing w on FA takes the path

$$1 \to \underbrace{3}_{\text{circuit 1}} \to 3 \to 2 \to 4 \to \underbrace{6}_{\text{circuit 2}} \to 6$$

- Bypassing all but the first circuit yields the path

$$1 \to \underbrace{3}_{\text{circuit 1}} \to 3 \to 2 \to 4 \to 6$$

  which corresponds to the word *bbaab*, which has length 5.

- However, we can go around the circuit one more time, yielding the path

$$1 \to \underbrace{3}_{\text{circuit 1}} \to \underbrace{3}_{\text{circuit 1}} \to 3 \to 2 \to 4 \to 6$$

  which corresponds to the word *bbbaab*, which has length 6.

Theorem 20 *There is an effective procedure to decide whether a given FA accepts a finite or an infinite language.*

Proof.

- Suppose that the FA has N states.
- Suppose that the alphabet consists of m letters.
- Then by Theorem 19, we only need to check all strings w with

  $$N \leq \text{length}(w) < 2N$$

  to determine if FA accepts an infinite language.
- If any of these are accepted, then the FA accepts an infinite language. Otherwise, it accepts a finite language.
- The number of strings w satisfying

  $$N \leq \text{length}(w) < 2N$$

  is

  $$m^N + m^{N+1} + m^{N+2} + \cdots + m^{2N-1}$$

  which is finite.
- Thus, checking all of these strings is an effective procedure.