CSCI 340: Computational Models

Decidability

Chapter 11

Department of Computer Science

The Close on the Theory of Automata

We have laid the foundation for the theory of finite automata.
We can represent these machines physically with electronics!
Unfortunately, we still have some unfinished business:

1. How can we tell whether two REs define the same language?
2. How can we tell whether two FAs accept the same language?
3. How can we tell whether the language defined by an FA has finitely many or infinitely many words in it (or any words at all)?

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**.

We need to provide decision procedures for problems we need to decide

Deciding Whether Two REs are Equivalent

Things to try:

- Generate words in lexicographical order *(no mathematical guarantee)*
- Interpret what regular expressions "mean" *(hopeless / not rigorous)*
- *what else?*

Deciding Whether Two REs are Equivalent

Things to try:

- Generate words in lexicographical order *(no mathematical guarantee)*
- Interpret what regular expressions "mean" *(hopeless / not rigorous)*
- *what else?*

We have already developed algorithms necessary for deciding the "equivalency problem" for FAs and therefore REs. We only need to recognize how to apply them.

Given two languages, L_1 and L_2, we can produce any of the following:

$$L_1' \qquad L_2' \qquad L_1 \cap L_2' \qquad L_2 \cap L_1'$$

We can produce an FA that accepts $(L_1 \cap L_2') + (L_2 \cap L_1')$.

If $L_1 = L_2$, then this FA cannot accept any words

Showing Equivalence – Method 1

1. Convert the FA into a regular expression
2. Delete all stars (*)
3. For each +, we throw away the right half and the +
4. When we have no more *'s or +'s we remove the parentheses
5. We now have a concatenation of a's, b's, and λ's

Showing Equivalence – Method 1

1. Convert the FA into a regular expression
2. Delete all stars (*)
3. For each +, we throw away the right half and the +
4. When we have no more *'s or +'s we remove the parentheses
5. We now have a concatenation of a's, b's, and λ's

Problem

We can't actually convert the FA representing $(L_1 \cap L_2') + (L_2 \cap L_1')$ *to a regular expression... but why?*

- *The last step when converting an RE to FA is "adding" together the edges from an initial state to a final state.*
- *When an FA doesn't accept anything, there will be no edges going from the initial to final state.*
- *We have reached a dead end by one of three ways: (1) no final state, (2) disconnected states, (3) unreachable final state from the start*

Showing Equivalence – Method 2

Examine the FA to see whether or not there is any path from initial to accepting. If there is any path, then the machine must accept some words. In a large FA with thousands of states, this may be impossible.

Try the following procedure:

1. Paint the start state blue
2. From each blue state, follow each edge that leads out of it and paint the destination state blue, then delete the edge followed
3. Repeat (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

At most, we will iterate through N times and remove at least $N - 1$ links (either all nodes are colored blue or we have already stopped)

Showing Equivalence – Method 2

Theorem

Let F be an FA with N states. Then if F accepts any words at all, it accepts some word with N or fewer letters.

Proof.

1. The shortest path from initial to accepting state (if any) cannot contain a circuit – otherwise it would no longer be the shortest path (cycles would occur otherwise)
2. If there is a path from initial to accepting then it can visit each state at most one time
3. The path can then have at most N edges and the word that generates it can have at most N letters.

□

Showing Equivalence – Method 3

- Test all words fewer than N letters by running them on the FA.
- If the FA accepts none of them, then it accepts no words at all.
- There are a predictable number of words to test, and each word takes a finite predictable time to run
- Therefore, this is an effective decision procedure

All three methods are **effective** – but which is the most efficient?

Example: $\mathbf{r_1} = \mathbf{a}^*$ $\mathbf{r_2} = \lambda + \mathbf{aa}^*$

Equivalence Theorem

We have proven the following (on prior slides):

Theorem

There is an effective procedure to decide whether:

1. *A given FA accepts any words*
2. *Two FAs are equivalent*
3. *Two regular expressions are equivalent*

go us!

Finiteness

Let us answer our last question of **decidability** – how can we tell whether an FA or regular expression defines a finite language or an infinite language

Regular expressions: if there is a closure of a non-empty set, then the language is infinite.

Finite automata: convert to a regular expression? OR...

Theorem

Let F be an FA with N states. Then:

1. *If F accepts a string $w \mid N \leq \text{length}(w) < 2N$, then F accepts an infinite language*
2. *If F accepts infinitely many words, then F accepts some word $w \mid N \leq \text{length}(w) < 2N$*

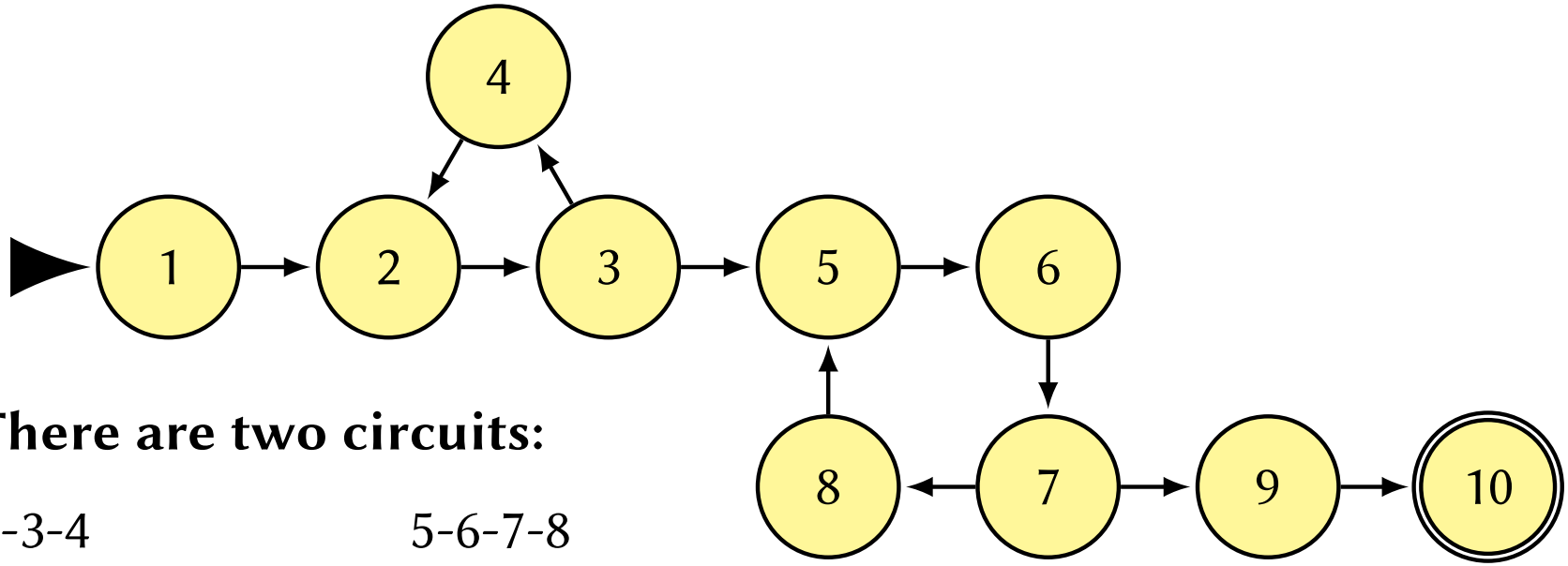
Finiteness

Proof.

1. A word is long enough to be pumped if it has more letters than the FA has states. If there is some word w with N or more letters, then we can split w into three parts $w = xyz$ and show that $xy^n z$ includes infinitely many words.
2. F must accept a word so large that its path must contain at least one circuit. Each circuit can contain at most N states. If we *bypass*, or leave the circuit as soon as we can, we will visit less than $2N$ letters. If we found a word less than size N, then we can increase it by taking the first circuit and looping once.

□

Finiteness Example



There are two circuits:

2-3-4 5-6-7-8

If we had a path: 1-2-3-4-2-3-5-6-7-8-5-6-7-8-5-6-7-9-10

and *bypassed* 5-6-7-8, we would be left with: 1-2-3-4-2-3-5-6-7-9-10

That path contains 11 states.

Procedure: We could test all strings between length N and $2N$ – if any are accepted, then the language is infinite.

Finiteness

Theorem

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

Proof.

If the machine has N states and the alphabet has m letters, then in total there are

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

different input strings in the range

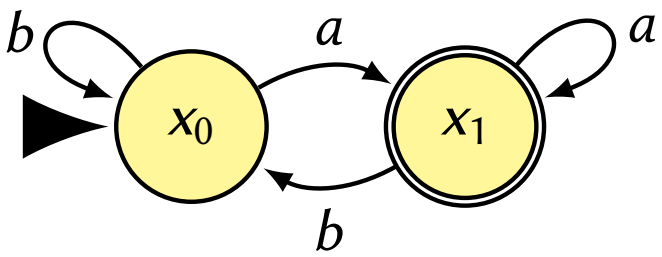
$$N \leq \text{length of string} < 2N$$

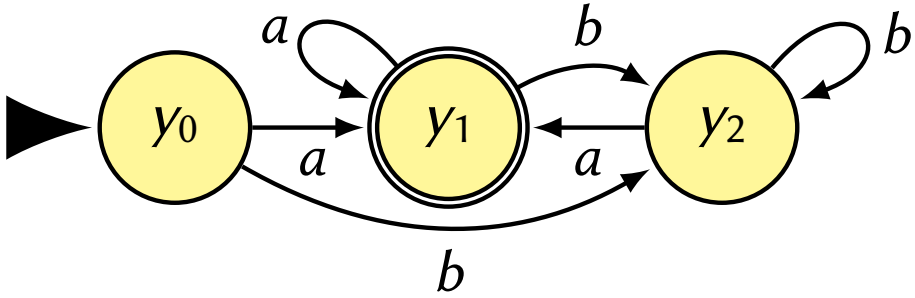
We can test them all by running them on the machine. If any are accepted, then the language is infinite. If none are accepted, the language is finite. □

Homework 6c

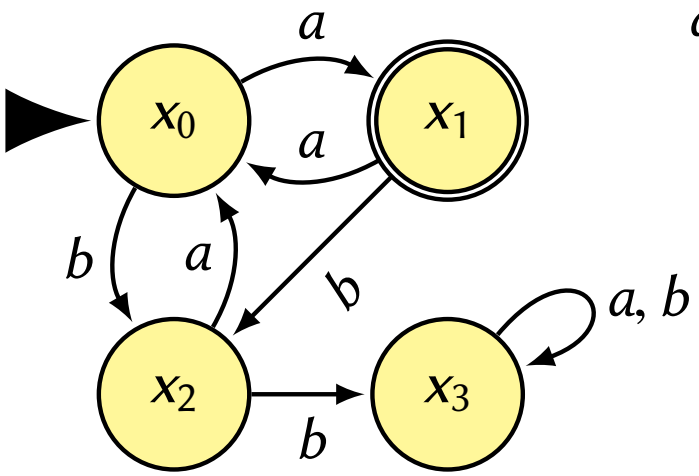
Show the following FAs are equivalent:

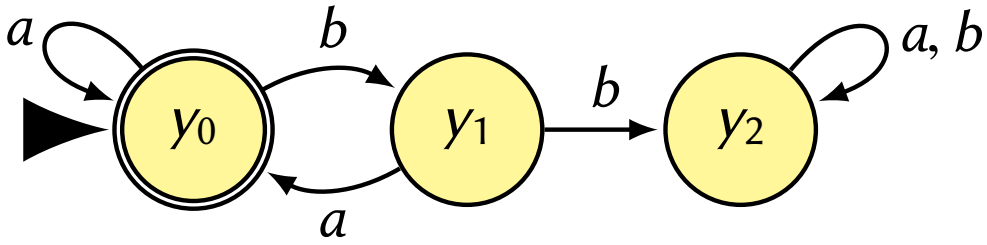
i





ii

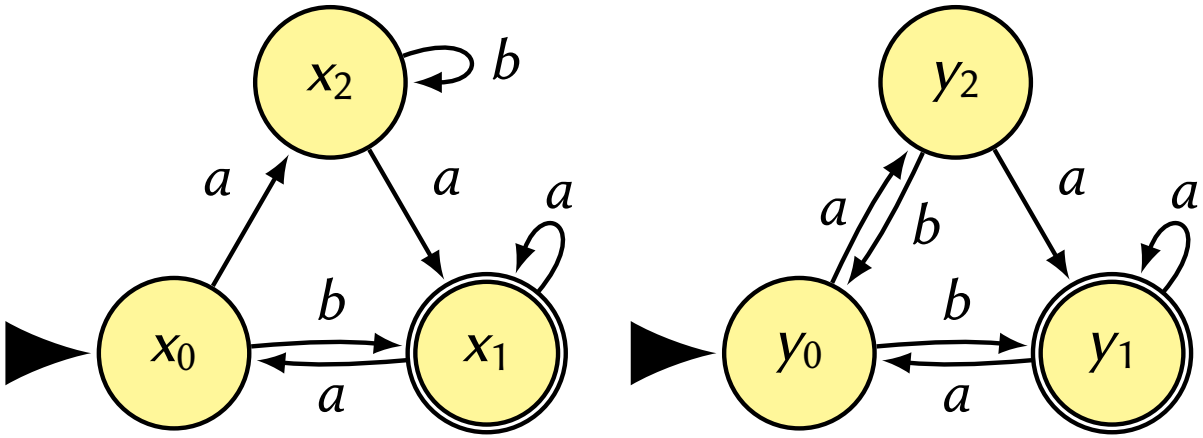




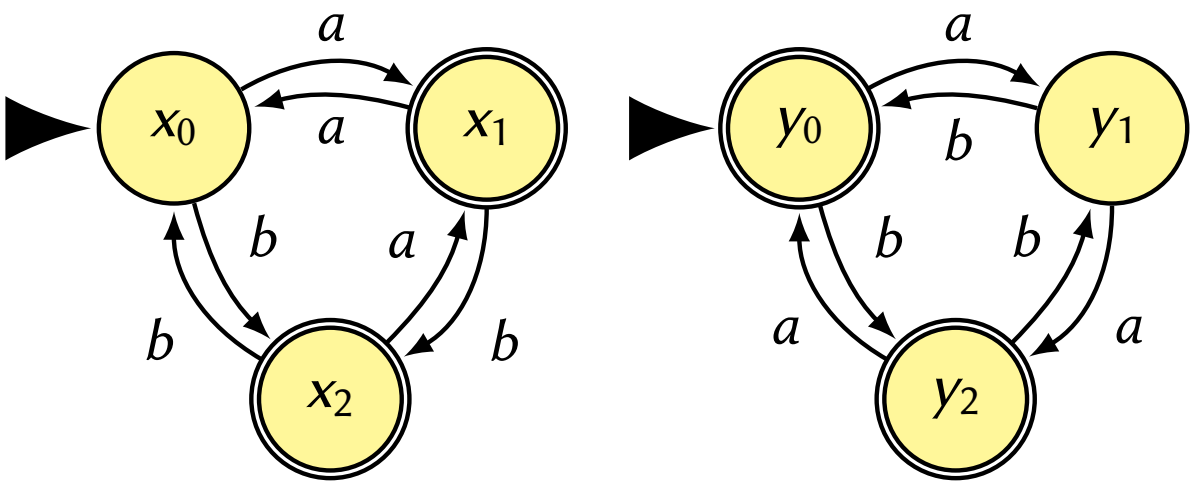
Homework 6c

Using the method of intersecting each machine with the complement of each other, determine whether two machines accept the same language (or not)

i



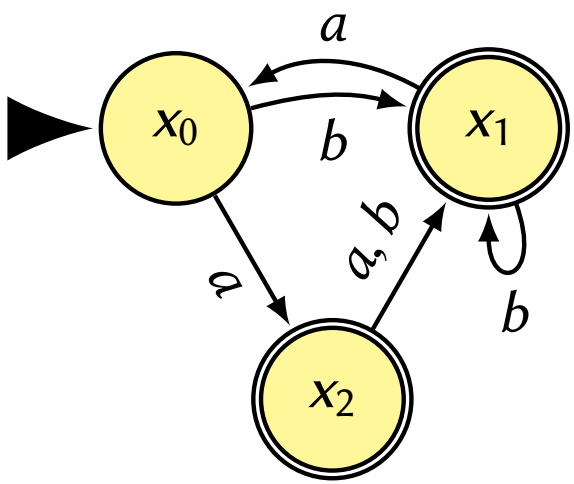
ii



Homework 6c

Do the following FAs accept a finite or infinite language? Justify.

i



ii