**Air Force Institute of Technology**

**Graduate School of Engineering and Management**

**Department of Electrical and Computer Engineering**

**CSCE 532 Automata and Formal Languages**

**Winter 2019**

# Day 11

# - The Church-Turing Thesis (cont.)

# - Decidability

§3.2 Variants of Turing Machines

### Terminology

A TM with the “stay-put” feature is a TM except that . If such a TM is in configuration and then the next configuration is .

### Theorem

A language is Turing-recognizable iff some TM with the “stay put” feature recognizes it.

#### Proof

Suppose is Turing-recognizable. Then it is recognizable by a TM M, which is a special case of a TM with the “stay-put” feature that never uses it. Now suppose is recognizable by a TM with the “stay-put” feature, and define the TM such that

* behaves exactly like whenever moves left or right; and
* whenever “stays put,” writes the same symbol as , moves right, rewrites the symbol under the head, and moves left.

Then .

### Terminology

A multitape TM is like an ordinary TM with tapes, each of which has its own read/write head. Initially tape 1 contains the input and the other tapes are blank. The transition function is

### Theorem

A language is Turing-recognizable iff some multitape TM recognizes it.

#### Proof

Suppose is Turing-recognizable. Then it is recognizable by a TM with the “stay-put” feature, which is a special case of a multitape TM. Now suppose is recognizable by a multitape TM . Then there exists a TM that simulates by

* storing the contents of its tapes on a single tape separated by a symbol not in ’s tape alphabet, and
* keeping track of the positions of ’s read/write heads using (a finite number of) additional symbols not in tape alphabet.

See the proof of Sipser’s Theorem 3.13 for additional details.

### Terminology

A nondeterministic TM is like an ordinary TM except that .

### Theorem

A language is Turing-recognizable iff some nondeterministic TM recognizes it.

#### Proof

Suppose is Turing-recognizable. Then it is recognizable by a TM, which is a special case of a nondeterministic TM. Now suppose is recognized by a nondeterministic TM . Then there exists a multitape TM that simulates by performing breadth first search on the tree of ’s possible configurations, accepting if it finds any accepting branch and rejecting if all branches terminate without accepting. See the proof of Sipser’s Theorem 3.16 for additional details.

### Terminology

An enumerator is a TM with an “attached printer” (a write-once device used as the primary output mechanism for many early digital computers). The language of an enumerator is the set of strings that it outputs.

### Theorem

A language is Turing-recognizable iff some enumerator enumerates it.

#### Unacceptable “Proof”

Suppose enumerates a language . Then a TM can recognize by comparing ’s output to its own input. Now suppose a TM recognizes a language . Then a TM can enumerate by generating every string in sequence and printing it iff accepts it.

### Practice (SIPSER Exercise 3.6)

Explain the flaw in the above “proof.”

#### Solution

In the second part, is not guaranteed to halt on inputs it does not accept, so can enter an infinite loop before it has printed all of the strings in . A correct proof adapted from Sipser follows.

#### Proof

Suppose enumerates a language . Then a TM can recognize by comparing ’s output to its own input. Now suppose a TM recognizes a language , let , and define the following TM:

“Ignore the input.

1. Repeat the following for
2. Run for steps on inputs
3. If any computation accepts, print the corresponding .”

Then every string in will be printed.

### Example (Sipser Problem 3.16a)

Show that the collection of Turing-recognizable languages is closed under union.

#### Solution

Proof: Let and be Turing-recognizable by and , respectively. Define

“On input :

1. If has not halted, run it one (more) step on . If it accepts, accept.
2. If has not halted, run it one (more) step on . If it accepts, accept.
3. If and have both halted, reject.
4. Go to step 1.

If accepts , it does so because either or accepts . Thus, . Conversely, if either or accepts , then it does so in a finite number of steps, in which case will accept in a finite number of iterations, each of which includes a finite number of steps. Thus, . Therefore .

### TEAM EFFORT (Sipser Problem 3.16b)

Show that the collection of Turing-recognizable languages is closed under concatenation.

#### Solution

# §3.3 The definition of Algorithm

* Other models of computation have been proposed (e.g. lambda calculus, -recursive functions, post systems, and rewriting systems)
* All have been shown to be equivalent (assuming finite work per step and finite steps)
* Gives confidence in the notion of an “algorithm” being well-defined
* Church-Turing Thesis: all possible (sufficiently broad) models of computation are equivalent.
* Terminology for defining TMs:
  + Formal description – states, transition functions, etc.
  + Implementation description – English prose to describe how TM moves its head and stores data on its tape
  + High-level description – English prose to describe an algorithm without talking about TM head and tape
* Sipser’s notation
  + Every mathematical object can and must be encoded (represented) as a string .
  + Until we start talking about complexity, we do not care *how* the object is encoded.
  + TM algorithms
    - Indented segment of text within quotes
    - First line describes input (e.g. or )
    - Broken into stages
    - Block structure indicated by indentation
    - TM implicitly tests any encoded inputs for validity

Example: Linz §§9.2 Exercise 2

Establish a convention for representing positive and negative integers in unary notation. With your convention, sketch the construction of a subtracter for computing .

Solution

Many possible conventions exist. One is to represent each integer as the difference between a first part and a second part . For example, could be represented by , , , etc., while -5 could be represented by , , etc. Subtraction can then be performed using only concatenation by observing that

For example, can be calculated from and as , which is .

### Practice: Sipser Exercise 3.7

Explain why the following is not a description of a legitimate TM.

“On input , a polynomial over variables :

1. Try all possible settings of to integer values.
2. Evaluate on all of these settings.
3. If any of these settings evaluates to , *accept*; otherwise, *reject*.”

#### Solution

We can never finish trying all possible settings, so it is not legitimate to say “otherwise, reject.”

### Practice: §§9.2 Exercise 6

Suggest a method for representing rational numbers on a TM, then [give a high level description] of a method for adding and subtracting such numbers.

#### Solution:

“On input , a pair of rational numbers:

1. Calculate and .
2. Calculate .”

What about negative numbers?!? Left as an exercise for the student.

### Practice:

Suggest a method for encoding DFAs.

#### Solution:

### Practice:

Suggest a method for encoding TMs.

#### Solution:

§4.1 Decidable Languages

* As we’ve already mentioned, there are problems that cannot be solved algorithmically.
* Recognizing such problems allows us to adjust our expectations and focus our attention on closely related problems that we can solve.
* It is convenient for our purposes to describe problems in terms of formal languages because we have developed the mathematical tools to reason about problems in that form.

### Example (Sipser Theorem 4.1)

Let .

#### Theorem

is decidable.

#### Solution

Note that is a set of strings, i.e. a language. Thus, we can ask the following questions, all of which are equivalent:

* Is is decidable?
* Is there a TM that decides ?
* Can we always determine whether or not a string is an element of ?
* Can we always determine whether or not a given DFA accepts a given string?

We already know the answer to the last question.

#### Proof

Let “On input , where is a DFA and is a string:

1. Simulate on input .
2. If the simulation ends in an accept state, *accept*. If it ends in a nonaccepting state, *reject*.”

Note that by definition of , regardless of whether or not , we assume that where is a DFA and is a string.

Now, by the definition of , if then is a DFA that accepts input string . Thus, if is started with as its input, stage 1 will end with in an accept state, and therefore will accept . In contrast, if then is a DFA that does not accept input string , stage 1 will end with in a nonaccepting state, and therefore will reject . We have shown that accepts all strings in and rejects all strings not in , i.e. decides . Therefore is decidable.

See Sipser for some implementation level details of (p. 195).

### Example (Sipser Theorem 4.2)

Let . Then is decidable.

#### Proof

Let “On input , where is an NFA and is a string:

1. Convert NFA to an equivalent DFA , using the procedure for this conversion given in Theorem 1.39.
2. Run TM from Theorem 4.1 on input .
3. If accepts, *accept*; otherwise *reject*.

Then decides .

### Example

The following languages are also decidable (see Sipser for proofs):