CMPSC 230
Theory of Computing
Fall 2008
Home page: `http://cs.allegheny.edu/wiki/cs230F2008/`

Notes from 10 October

Review

We have seen the general description of Turing machines that can behave as language recognizers (much as DFAs recognize regular languages and PDAs recognize context-free languages). The most important concept here is that such a machine, given an input string written on its tape, must eventually halt in either a state named q_{accept}, meaning that it "recognizes" the string as an element of a given language, or in a state named q_{reject}, meaning that the string is not an element of the language.

Given any language whatsoever, can we build a Turing machine that can recognize that language? Usually we use the word "decide" rather than "recognize," and we call a language *decidable* if there is some Turing machine that decides whether or not a given string belongs to that language.

In the next chapter in the book we will construct a language that is not decidable, but for now let's just look at the problem from a more abstract point of view.

How Many Turing Machines Are There?

[This has been somewhat simplified from the presentation in class.]

Let's agree on a way to assign a unique integer to every symbol that could possibly be used as a symbol on a Turing machine tape. If we just assign a unique number to every character and digit in every written language on Earth, that ought to do it. We'll make the "blank" symbol $\sqcup$ always "0" since every Turing machine must have a blank symbol. Just for the sake of argument, suppose our input alphabet $\Sigma = \{a, b\}$ and we choose the representation $a \leftrightarrow 1$ and $b \leftrightarrow 2$.

Similarly, let's agree that we will always represent states by integers, and furthermore, let's agree that the start state, q_0, will always be represented by "0", the accept state q_{accept} will always be represented by "1", the reject state q_{reject} will always be represented by "2", and all other states will be represented by successively larger integers. Finally, let's agree to represent the direction "left" or "L" by the integer 1 and the direction "right" or "R" by 2.

We can now describe a Turing machine by describing its transition function δ as a sequence of integers, grouped into fives. Here's an example:

> **Turing machine:** $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where $Q = \{q_0, q_{reject}, q_{accept}\}$, $\Sigma = \{a, b\}$, $\Gamma = \{a, b, \sqcup\}$, and
>
> $$\begin{aligned} \delta(q_0, a) &= (q_{accept}, a, R) \\ \delta(q_0, b) &= (q_0, b, R) \\ \delta(q_0, \sqcup) &= (q_{reject}, \sqcup, R) \end{aligned}$$

This machine decides the language $L = \{w \mid w \text{ contains at least one } a\}$. Here is its encoding as a sequence of integers:

```
0, 1, 1, 1, 2; 0, 2, 0, 2, 2; 0, 0, 2, 0, 2
```

We don't really need the semicolons, since each group of five integers defines the five parts of every transition.

Any Turing machine can be described by such a sequence of integers.

Unique Factorization

Before we go on, we recall a fact from mathematics: every positive integer greater than 1 can be uniquely written as a product of prime numbers. For example, the number 756 can be written as $2 \times 2 \times 3 \times 3 \times 3 \times 7 = 2^2 \times 3^3 \times 7$.

Furthermore, there are infinitely many prime numbers. We will use this fact to show that every Turing machine can be represented by an integer.

Gödel Numbers

List all the prime numbers in increasing order:

$$2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, \ldots$$

Given a numerical description of a Turing machine, we can convert it into a single integer by using the numbers as exponents for a sequence of primes. In the example above, the encoding would be:

$$2^0 \times 3^1 \times 5^1 \times 7^1 \times 11^2 \times 13^0 \times 17^2 \times 19^0 \times 23^2 \times 29^2 \times 31^0 \times 37^0 \times 41^2 \times 43^0 \times 47^2$$

This notion of assigning an integer to each element of an infinite set of strings (in this case, strings of integers) was described by mathematician Kurt Gödel and was used in the proof of his famous "Incompleteness Theorem." (Gödel was a giant of twentieth-century mathematics — look him up. For instance, in 1999 Douglas Hofstadter wrote a nice description of Gödel's work for *Time* magazine.)

Thus, every Turing machine corresponds to an integer; several, actually, since we can re-order the list of transistions to produce different Gödel numbers for the same machine.

Does every integer correspond to a Turing machine? Sure, if we agree that any integer whose prime factorization doesn't give a sequence of exponents that "make sense" can be identified with a trivial "do-nothing" Turing machine. We have just shown that we can match every Turing machine up with an integer.

The Set of All Languages

How many languages are there? Can we match each language up with an integer? Suppose we can. Let's create a table. Each row of the table is an integer, representing a Turing machine (as just described above). Each column represents a string over some alphabet, say $\{a, b\}$. It should be obvious that we can list all of the strings in some predefined order, e.g., by length, and alphabetically for each length. Each entry of the table is either "yes" or "no". If we look in row i, column w, the entry tells us whether the i-th Turing machine accepts or rejects the string w. Obviously each row corresponds to a language, namely, the language of all strings that have a "yes" entry in that row.

Is there a language that does not appear in any row? Yes — we will construct it. We want a language that is not accepted by any Turing machine, so we will make sure that every Turing machine in the list either accepts a word that is not in this language, or rejects a word that is in this language.

Let's start with the first word, ε, and look at the first Turing machine in the list. If that Turing machine accepts ε, don't include it in the new language, otherwise include it. We now know that the language is not decided by the first Turing machine. Look at the second word, a, and look at the second Turing machine. Again, if this Turing machine accepts a, don't include a, otherwise include it. Now our language is not decided by either of the first two Turing machines.

We precede in this way for each word and Turing machine. If the i-th word is accepted by the i-th Turing machine, don't include it in the language, else include it. In this manner we construct a language that is not decided by any Turing machine in the list. We have an undecidable language.

This is called a "diagonalization" argument. The same argument is used to prove (in Math 205??) that there are "more" real numbers than there are integers.

We have:

Theorem. There is a language that can not be decided by any Turing machine.