MA/CSSE 474

Theory of Computation

Decision Problems

Languages, Machines, Computation

Decision Problems

A ***decision problem*** is simply a problem for which the answer is yes or no (True or False). A ***decision procedure*** answers a decision problem.

Examples:

- Given an integer *n*, does *n* have a pair of consecutive integers as factors?

- The language recognition problem: Given a language *L* and a string *w*, is *w* in *L*?

↑

Our focus in this course

The Power of Encoding

Anything can be encoded as a string.
For example, on a computer everything is encoded as a string of bits.

<*X*> is the string encoding of *X*.
<*X*, *Y*> is the string encoding of the pair *X*, *Y*.

Problems that don't look like decision problems about strings and languages can be recast into new problems that do look like that.

Web Pattern Matching

Pattern matching on the web:

- Problem: Given a search string *w* and a web document *d*, do they "match"? In other words, should a search engine, on input *w*, consider returning *d*?
- The language to be decided:

{<*w*, *d*> : *d* is a candidate match for the string *w*}

The Halting Problem

Does a program always halt?

- Problem: Given a program *p*, written in some some standard programming language L, is *p* guaranteed to halt, no matter what input it is given?

- The language to be decided:

$$HP_{ALL} = \{p \in L : p \text{ halts on all inputs}\}$$

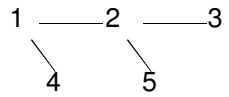
Primality Testing

- **Problem**: Given a nonnegative integer *n*, is it prime?
- An **instance** of the problem: Is 9 prime?
- To encode the problem we need a way to encode each instance: We encode each nonnegative integer as a binary string.
- The language to be decided:

PRIMES = {*w* : *w* is the binary encoding of a prime integer}.

Graph Connectivity

- Problem: Given an undirected graph G, is it connected?
- Instance of the problem:



- Encoding of the problem: Let V be a set of binary numbers, one for each vertex in G. Then we construct $\langle G \rangle$ as follows:
  - Write $|V|$ as a binary number,
  - Write a list of edges,
    Each pair of binary numbers represents one edge.
  - Separate all such binary numbers by "/".

    1/10/1/100/10/101/10/11
- The language to be decided: CONNECTED = $\{w \in \{0, 1, /\}^* : w = n_1/n_2/\ldots n_i$, where each n_i is a binary string and w encodes a connected graph, as described above$\}$.

Turning Problems Into Decision Problems

Casting multiplication as decision:

- Problem: Given two nonnegative integers, compute their product.
- Encoding of the problem: Transform computation into verification.
- The language to be decided, INTEGERPROD

  $\{w$ of the form:
  $<integer_1>$x$<integer_2>=<integer_3>$, where:
  $<integer_n>$ is any well formed integer, and
  $integer_3 = integer_1 * integer_2\}$

  `12x9=108` $\in$ INTEGERPROD
  `12=12` $\notin$ INTEGERPROD
  `12x8=108` $\notin$ INTEGERPROD

Turning Problems Into Decision Problems

Casting sorting as decision:

- Problem: Given a list of integers, sort it.
- Encoding of the problem: Transform the sorting problem into one of examining a pair of lists.
- The language to be decided:

$L = \{w_1 \# w_2 : \exists n \geq 1$
(w_1 is of the form $<int_1, int_2, \ldots int_n>$,
w_2 is of the form $<int_1, int_2, \ldots int_n>$, and
w_2 contains the same objects as w_1 and
w_2 is sorted)$\}$

Examples:

`<1,5,3,9,6>#<1,3,5,6,9>` $\in L$
`<1,5,3,9,6>#<1,2,3,4,5,6,7>` $\notin L$

The Traditional Problems and their Language Formulations are Equivalent

By equivalent problems, we mean that either problem can be ***reduced to*** the other.

What does it mean to reduce Problem A to Problem B?
If we have a machine to solve B, we can use it to build a machine solve A, using only the B and other functions that can be constructed using machines of equal or lesser power than the B machine.

Note: Reduction does not always preserve efficiency!

An Example

Consider the multiplication example: INTEGERPROD

{*w* of the form:

$<integer_1> \times <integer_2> = <integer_3>$, where:

$<integer_n>$ is any well formed integer, and

$integer_3 = integer_1 * integer_2$ }

Given a multiplication machine, we can build the language recognition machine:

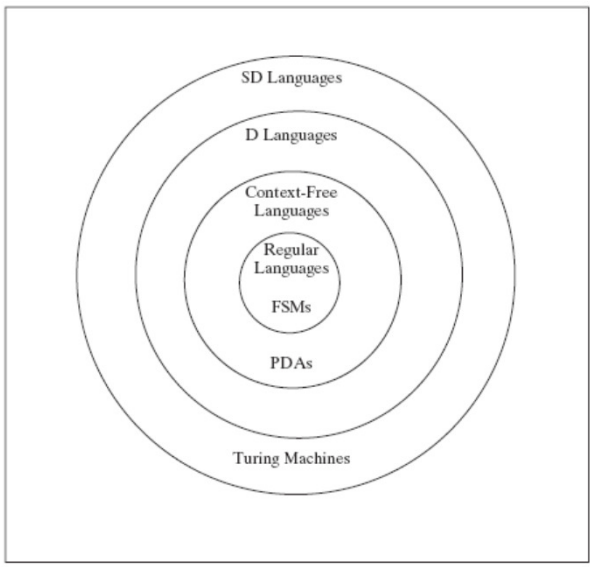
Given the language recognition machine, we can build a multiplication machine:

Q1-2

Languages and Machines

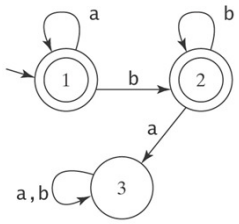
Quick overview today.

The details will comprise most of the rest of the course.



Finite State Machines

An FSM to accept a*b*:

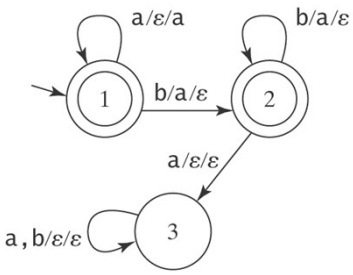


An FSM to accept $A^nB^n = \{a^n b^n : n \geq 0\}$

Q3

Pushdown Automata

A PDA to accept $A^nB^n = \{a^n b^n : n \geq 0\}$



Example: aaabb

Stack:

Other Examples

Bal, the language of balanced parentheses.

PalSpecial: $\{wcw^R : w \in \{a,b\}^*\}$

PalEven: $\{ww^R : w \in \{a,b\}^*\}$

Q4

Trying Another PDA

A PDA to accept strings of the form:

$A^nB^nC^n = \{a^n b^n c^n : n \geq 0\}$

Turing Machines

A read/write head and a tape that is infinite in both directions.

Based on current state and symbol it sees on the tape, it writes a symbol (possibly the same one) to the tape and moves one position left or right.

If it reaches an accepting state or rejecting state, it halts.

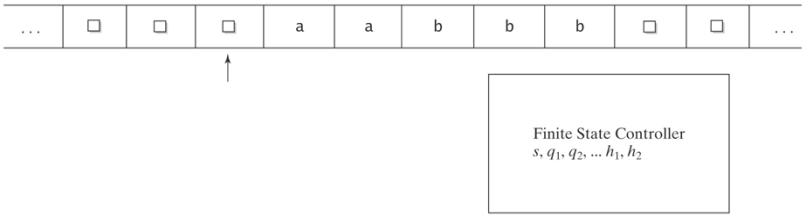
The (finite) input string is originally written consecutively on a portion of the tape; the rest is initially blank, but the read/write head can write things on any square.

At any given time, only a finite part of the tape is non-blank.

The head starts at the square to the left of the first input symbol.

Turing Machines

A Turing Machine to accept $A^nB^nC^n$:



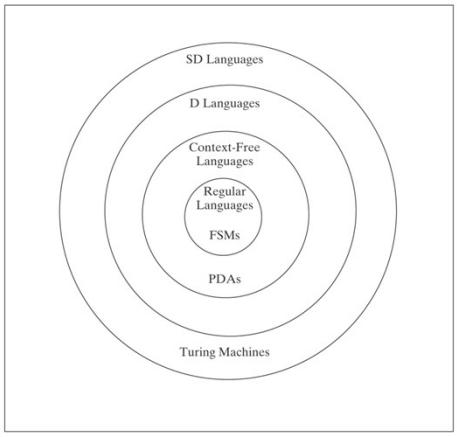
Q5

Decidable and Semidecidable Languages

- A language L is **decidable** iff there exists a Turing machine M that halts on all inputs, accepts all strings that are in L, and rejects all strings that are not in L.
  - In other words, M can always say yes or no, as appropriate.
- A language L is **semidecidable** iff there exists a Turing machine M that accepts all strings that are in L and fails to accept every string that is not in L.
  - Given a string that is not in L, M may reject or it may loop forever.
  - In other words, M can always recognize a string in L and say yes,
  - but it may not know when it should give up looking for a solution and say no.
- A language L is **undecidable** iff it is not semidecidable.

Q6

Languages and Machines



Rule of Least Power: "Use the least powerful language suitable for expressing information, constraints or programs on the World-wide web."
--Tim Berners-Lee and Noah Mendelsohn(2006)

Some "Canonical" Languages

- $A^nB^n = \{a^nb^n : n >= 0\}$
- Bal = { strings of balanced parentheses}
- $WW = \{ww : w \in \Sigma^*\}$
- PalEven $\{ww^R : w \in \Sigma^*\}$
- $A^nB^nC^n = \{a^nb^nc^n : n >= 0\}$
- HP_{ALL} = {<T> : T is a Turing machine that eventually halts, no matter what input it is given}
- PRIMES = {*w* : *w* is the binary encoding of a prime integer}

Nondeteminism

- A **nondeterministic** machine in a given state, looking at a given symbol (and with a given symbol on top of the stack if it is a PDA), has a choice of multiple possible moves that it can make.
- If there is a move that leads toward acceptance, it makes that move.

Nondeterminism

- Given a string in {a, b}*, is it in
  PalEven = $\{ww^R : w \in \{a,b\}^*\}$?
- PDA
- **Choice:** Continue pushing, or start popping?
- This language can be accepted by a nondeterministic PDA but not by any deterministic one.

Q7

Nondeterministic value-added?

- Ability to recognize additional languages?
  - FSM: no
  - PDA : yes
  - TM: no

We will prove these later

- Ease of designing a machine for a particular language
  - Yes in all cases

Nondeterministic Computation

1. *choose* (action 1;;
   action 2;;
   …
   action *n*)

First case: Each action will return True, return False, or run forever.

If any of the actions returns TRUE, choose returns true.
If all of the actions return FALSE, choose returns FALSE.
If none of the actions return TRUE, and some do not halt, choose does not halt.

2. *choose*(*x* from *S*: *P*(*x*))

Second case: S may be finite, or infinite with a generator (enumerator).

If P returns true on some x, so does Choose
If it can be determined that P(x) is false for all x in P, return False.
Otherwise, fail to halt.