MA/CSSE 474

Theory of Computation

Decision Problems, Continued
DFSMs

Your Questions?

- Friday's class
- Reading Assignments
- HW1 solutions
- HW2 or HW3
- Anything else

Recap: Turning Problems into Language Recognition Problems

Cast multiplication as language recognition:

- Problem: Given two nonnegative integers, compute their product.
- Encode the problem: Transform computing into verification.
- The language to be decided:

INTEGERPROD = {w of the form:
<int_1>x<int_2>=<int_3>, where each <int_n> is an encoding (decimal in this case) of an integer, and $int_3 = int_1 * int_2$}

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

Recap: Show the Equivalence

Consider the multiplication language example:
INTEGERPROD = {w of the form:
<int_1>x<int_2>=<int_3>, where each <int_n> is an encoding (decimalin this case) of an integer, and
$int_3 = int_1 * int_2$}

Given a multiplication function for integers, we can build a procedure that recognizes the *INTEGERPROD* language: **(easy, we did it last time)**

Given a function *R*(*w*) that recognizes *INTEGERPROD*, we can build a procedure *Mult*(*m*,*n*) that computes the product of two integers: **(you were supposed to figure this out during the weekend)**

Regular Languages (formally)

More on Finite State Machines

Recap - Definition of a DFSM

$M = (K, \Sigma, \delta, s, A)$, where:

The D is for Deterministic

K is a finite set of ***states***

Σ is a (finite) ***alphabet***

$s \in K$ is the ***initial state*** (a.k.a. start state)

$A \subseteq K$ is the set of ***accepting states***

$\delta: (K \times \Sigma) \rightarrow K$ is the ***transition function***

Sometimes we will put an M subscript on K, Σ, δ, s, or A (for example, s_M), to indicate that this component is part of machine M.

Acceptance by a DFSM

$M = (K, \Sigma, \delta, s, A)$

Informally, M ***accepts*** a string w iff M winds up in some element of A after it has finished reading w.

The ***language accepted by M***, denoted $L(M)$, is the set of all strings accepted by M.

But we need more formal notations if we want to prove things about machines and languages.

Today we examine the book's notation, ⊢. Unicode 22A2. That symbol is commonly called *turnstile* or *tee*. It is often read as "derives" or "yields"

Configurations of a DFSM

A ***configuration*** of a DFSM M is an element of:

$$K \times \Sigma^*$$

It captures the two things that affect M's future behavior:

- its current state
- the remaining input to be read.

The ***initial configuration*** of a DFSM M, on input w, is:

$$(s_M, w)$$

Where s_M is the start state of M.

The "Yields" Relations

The ***yields-in-one-step*** relation: $\vdash_M$:

$(q, w) \vdash_M (q', w')$ iff

- $w = a\,w'$ for some symbol $a \in \Sigma$, and
- $\delta(q, a) = q'$

The ***yields-in-zero-or-more-steps*** relation: $\vdash_M{}^*$

$\vdash_M{}^*$ is the reflexive, transitive closure of $\vdash_M$.

Note that this accomplishes the same thing as the "extended delta function" that we considered on Day 1. Two notations for the same concept.

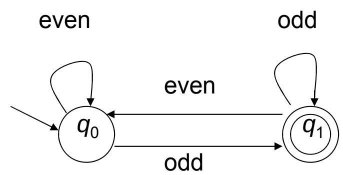
Computations Using FSMs

A ***computation*** by M is a finite sequence of configurations $C_0, C_1, \ldots, C_n$ for some $n \geq 0$ such that:

- C_0 is an initial configuration,
- C_n is of the form (q, ε),
  for some state $q \in K_M$,
- $\forall i \in \{0, 1, \ldots, n-1\}\ (C_i \vdash_M C_{i+1})$

An Example Computation

A FSM M that accepts decimal representations of odd integers:



On input 235, the configurations are:

$(q_0, 235) \vdash_M (q_0, 35)$
$\vdash_M (q_1, 5)$
$\vdash_M (q_1, \varepsilon)$

Thus $(q_0, 235) \vdash_M{}^* (q_1, \varepsilon)$

Accepting and Rejecting

A DFSM M ***accepts*** a string w iff:

$(s_M, w) \vdash_M{}^* (q, \varepsilon)$, for some $q \in A_M$

A DFSM M ***rejects*** a string w iff:

$(s_M, w) \vdash_M{}^* (q, \varepsilon)$, for some $q \notin A_M$

The ***language accepted by*** M, denoted $L(M)$, is the set of all strings accepted by M.

Theorem: Every DFSM M, in configuration (q, w), halts after $|w|$ steps.

Thus every string is either accepted or rejected by a DFSM.

Proof of Theorem

Theorem: Every DFSM *M*, in configuration (q, w), halts after |*w*| steps.

Proof: by induction on |w|

Base case: n = 0, so w is ε, it halts after 0 steps.

Induction step: Assume true for strings of length n and show for strings of length n+1.

Let $w \in \Sigma^*$, $w \neq \varepsilon$. Then $|w| = n+1$ for some $n \in \mathbb{N}$.

So w must be au for some $a \in \Sigma$, $u \in \Sigma^*$, $|u| = n$.

Let q' be $\delta(q, a)$. By definition of $\vdash$, $(q,w) \vdash_M (q', u)$

By the induction hypothesis, starting from configuration (q', u), M halts after n steps.

Thus, starting from the original configuration, M halts after n+1 steps.

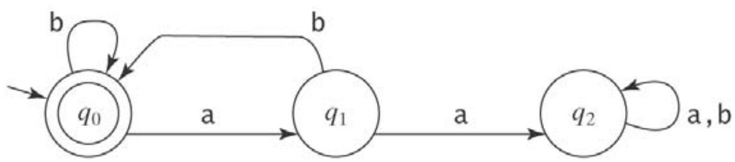
Regular Language Formal Definition

Definition:
A language L is ***regular***
iff
L = L(M) for some DFSM M

Example

$L = \{w \in \{a, b\}^* :$

every a is immediately followed by a b$\}$.



δ can also be represented as a **transition table**:

state ↓ input →	a	b
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_2

q_2 is a **dead state**.

Exercises: Construct DFSMs for

$L = \{w \in \{0, 1\}^* : w$ has odd parity$\}$.
I.e. an odd number of 1's.

$L = \{w \in \{a, b\}^* :$

no two consecutive characters are the same$\}$.

$L = \{w \in \{a, b\}^* : \#_a(w) >= \#_b(w) \}$

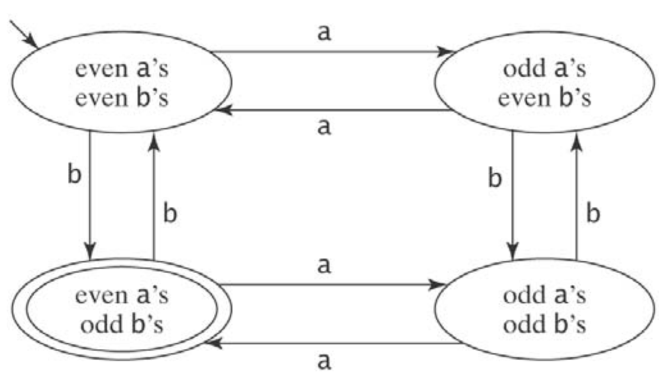
$L = \{w \in \{a, b\}^* : \forall x,y \in \{a, b\}^* \ (w=xy \rightarrow |\#_a(x) - \#_b(x)| <= 2) \}$

MORE DFSM EXAMPLES

Examples: Programming FSMs

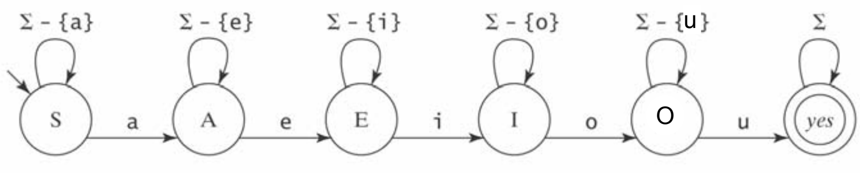
Cluster strings that share a “future”.

$L = \{w \in \{a, b\}^* : w$ contains an even number of a’s and an odd number of b’s$\}$



Vowels in Alphabetical Order

$L = \{w \in \{a - z\}^* :$ all five vowels, a, e, i, o, and u, occur in w in alphabetical order$\}$.

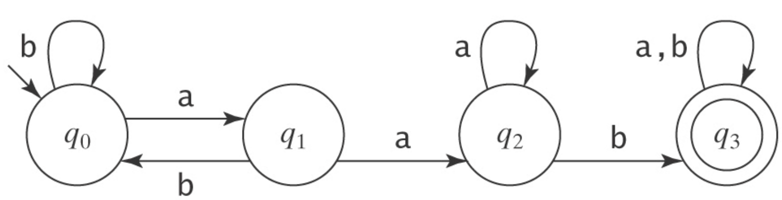


THIS EXAMPLE MAY BE **facetious**!

Negate the condition, then …

$L = \{w \in \{a, b\}^* : w$ does not contain the substring aab$\}$.

Start with a machine for the complement of L:



How must it be changed?

The Missing Letter Language

Let Σ = {a, b, c, d}.

Let $L_{Missing}$ =
{$w \in \Sigma^*$: there is a symbol $a_i \in \Sigma$ that does not appear in w}.

Try to make a DFSM for $L_{Missing}$:

Expressed in first-order logic,
$L_{Missing}$ = $\{w \in \Sigma^* : \exists a \in \Sigma\ (\forall x,y \in \Sigma^* (w \neq xay))\}$

NONDETERMINISM

Nondeterminism

- 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), may have a choice of several possible moves that it can make.
- If there is a move that leads toward acceptance, it makes that move.

Necessary Nondeterminism?

- As you saw in the homework, a PDA is a FSM plus a stack.
- 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.

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

A Way to Think About 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, *choose* returns FALSE.
Otherwise, *choose* fails to halt.