Languages

January-08-13 10:51 AM

Symbol

Primitive notion
Example:
0, 1, 2,
a, b, c

Alphabet

Finite nonempty set of symbols
Example:
$\Sigma = \{0, 1\}$
$\Sigma = \{a, b, c, \dots, z\}$

String (Word)

A finite sequence of symbols.
Example:
abca
ϵ - empty string (sometimes λ, Λ)

Length

$|x|$ - length of string x
$|\epsilon| = 0$

Star

Σ^* - set of all finite strings over Σ

$\Sigma = \{0, 1\}$
$\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\}$

Concatenation

$WX = W \cdot X$

$W =$ case
$X =$ book
$XW =$ bookcase
$WX =$ casebook

Concatenation no commutative in general, but it is associative.
$(xy)z = x(yz)$

String Subsets

A string y is a **prefix** of x if $\exists z$ such that $x = yz$
A string z is a **suffix** of x if $\exists y$ such that $x = yz$
A string t is a **substring** of x if $\exists y, z$ such that $x = ytz$

Exponentiation

$xx = x^2$
$x \cdots x = x^n, \qquad n \geq 0$
$x^1 = x$
$x^0 = \epsilon$
$x^{a+b} = x^a x^b$
Note:
$(xy)^n \neq x^n y^n, \qquad$ in general

Counting Occurrences

$|x|_a = \#$ of occurrences of the letter a in the work x
$|\text{cabbage}|_a = 2$

Reversing

$x^R =$ reversal of the word x
$(\text{desserts})^R = "stressed$

Palindrome

$x = x^R$

Formalism of Reversal

$$w^R = \begin{cases} \epsilon, & \text{if } w = \epsilon \\ ax^R, & \text{if } w = xa, a \in \Sigma \end{cases}$$

Recursive definition of reversal

Theorem

$(xy)^R = y^R x^R$
for strings x, y

Proof

By induction on the length of $|y|$

Base case: $|y| = 0 \Rightarrow y = \epsilon$
$(xy)^R = x^R$
$y^R x^R = x^R$

Induction:
Assume true for $|y| = n$. Prove for $|y| = n + 1$
$(xy)^R$
$y = za, \qquad a \in \Sigma$
$y^R = az^R$
$(xy)^R = (xza)^R = a(xz)^R = az^R x^R = y^R x^R$

Open Problem

Start with arbitrary string: 22323
Look for number of repetitions. In above example have 2323. Write down order of repetition: 223232
2232322
22323222
223232223
2232322231

Conjecture: No matter what finite string you start with you eventually reach 1.

Language

A language L over an alphabet Σ is a subset of Σ^*

Union

$L_1 \cup L_2$

Intersection

$L_1 \cap L_2$

Complement

$\bar{L} = \Sigma^* - L$

Examples of Languages

$PAL = \{x \in \{a, b\}^* : x = x^R\} = \{\epsilon, a, b, aa, bb, aaa, aba, \dots\}$

$\{x \in \{a, b\}^* : |x|_a = |x|_b\} = \{\epsilon, ab, ba, aabb, abab, \dots\}$

Example of Language Concatenation

$L_1 = \{\text{over}, \text{under}\}$
$L_2 = \{\text{salted}, \text{worked}\}$
$L_1 L_2 = \{\text{overstaffed}, \text{overworked}, \text{understaffed}, \text{underworked}\}$

Special Languages
$\emptyset$ - empty set
Σ^* - all strings

Normal properties of sets apply
$L_1 \cup (L_2 \cap L_3) = (L_1 \cup L_2) \cap (L_1 \cup L_3)$
De Morgan's laws
$\overline{L_1 \cup L_2} = \overline{L_1} \cap \overline{L_2}$

Concatenation
$L_1 L_2 = L_1 \circ L_2 = \{xy : x \in L_1, y \in L_2\}$
$L^n = L \circ L \circ \cdots \circ L$ (n times)
$L^0 = \{\epsilon\}$
$L^{m+2} = L^m \circ L^n$
$L^1 = L$
$$L^n = \begin{cases} \{\epsilon\} & \text{if } n = 0 \\ L \circ L^{n-1} & \text{if } n \geq 1 \end{cases}$$

Kleene *
$$L^* = \bigcup_{n \geq 0} L^n = L^0 \cup L^1 \cup L_2 \cup \cdots$$
Also known as **Kleene Closure** since $(L^*)^* = L^*$

Positive Closure
$$L^+ = LL^* = \bigcup_{n \geq 1} L^n$$

Theorem
If $L^2 \subseteq L$ then $L^+ \subseteq L$

Proof of Theorem
Equivalent Statement:
$$If\ L^2 \subseteq L \text{ then} \bigcup_{n \geq 1} L^n \subseteq L$$
And Equivalently
If $L^2 \subseteq L$ then $L^n \subseteq L$ forall $n \geq 1$ (*)

Proof by induction on n.
Base case: $n = 1, 2$
$n = 1$: $L^1 = L \subseteq L$
$n = 2$: $L^2 \subseteq L$ by hypothesis

Induction: Assume (*) is true for n and prove it for $n + 1$
Assume $L^n \subseteq L$
By homework #1, know that $A \subseteq B \Rightarrow LA \subseteq LB$

$L \circ L^n \subseteq L \circ L$
$L^{n+1} \subseteq L^2 \subseteq L$ by hypothesis
$\Rightarrow L^{n+1} \subseteq L$

Regular Expressions and DFAs

January-10-13 10:23 AM

Regular Expressions - Kleene (1956)

- A way to specify languages
- A regular expression is a string over the alphabet
  $\Delta = \{\cup, *, (,)\} \cup \Sigma \cup \{\epsilon, \emptyset\}$
  In this case $\emptyset$ and ϵ are symbols (not empty string/set, but they represent empty string/set in the regular expression language)

$L(r)$ = the language represented by a regular expression

$L(\epsilon) = \{\epsilon\}$
$L(\emptyset) = \emptyset$
$L(a) = \{a\}, \quad a \in \Sigma$
$L(r_1 \cup r_2) = L(r_1) \cup L(r_2)$
$L((r_1)^*) = L(r_1)^*$
$L((r_1)(r_2)) = L(r_1)L(r_2)$

Extra parentheses can be removed. Ex: $((a)(b)) \rightarrow ab$

Precedence

1. *
2. Concatenation (implicit)
3. $\cup$

Language Classes

REG = { L : L specified by a regular expression }
= the collection of regular languages

FINITE = { L : L has finitely many elements}

FINITE $\subsetneq$ REG

Theorem

The class of regular languages is closed under

- union
- concatenation
- Kleene *

L_1 reg, L_2reg $\Rightarrow L_1 \cup L_2, L_1 L_2, L_1^* \subseteq$ REG

How about intersection or complement of languages?

Deterministic Finite Automaton (DFA)

- Accepters or recognizers of languages

McCulloch & Pits 1943

Specification

Q = a finite nonempty set of states often written as $\{q_0, q_1, \ldots, q_{n-1}\}$
Σ = an alphabet
$F \subseteq Q$, the set of accepting states (final states)
$q_0 \in Q$, the initial or start state
$\delta: Q \times \Sigma \rightarrow Q$, transition function

A DFA is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$ where the pieces are as above.

Extended Transition Function

$\hat{\delta}: Q \times \Sigma^* \rightarrow Q$
$\hat{\delta}(q, w)$ = the state I end up in if, starting in state q, I read the input w

Recursive Definition

$\hat{\delta}(q, \epsilon) = q$
$\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a), \qquad w \in \Sigma^*, a \in \Sigma$

Theorem

$\hat{\delta}(q, a) = \delta(q, a)$
So the ^ is usually omitted from $\hat{\delta}$

Language Acceptance (Recognition)

The language **accepted by** $M = (Q, \Sigma, \delta, q_0, F)$ (**recognized by**) is $L(M) = \{x \in \Sigma^* : \delta(q_0, x) \in F\}$

Theorem

Let L_1, L_2 be languages over Σ accepted by DFA's M_1 and M_2, respectively. Then there is a DFA accepting $L = L(M_1) \cap L(M_2)$

Problem Set

Find a regular expression for $L_n = \{x \in \{1, 2, \ldots, n\}^* : \text{x contains no two consecutive equal symbols}\}$
$L_3 = \{1, 2, 3, 12, 13, 21, 23, 31, 32, 121, \ldots\}$

Examples of Regular Expressions

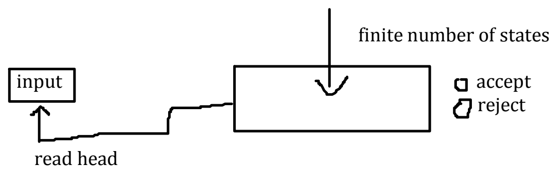
All strings over $\{a, b\}$ of length < 3
$(a \cup b \cup \epsilon)(a \cup b \cup \epsilon)(a \cup b \cup \epsilon)$

All strings over $\{a, b\}$ having aa as a substring.
$(a \cup b)^* aa (a \cup b)^*$

All strings over $\{a, b\}$ not having aa as a substring
$b^*(abb^*)^*(a \cup \epsilon)$
$(a \cup \epsilon)(b \cup ba)^*$

Strings over $\{a, b, c\}$ in which all a's precede all b's which precede all c's
$a^* b^* c^*$
The same as above but remove ϵ
$aa^* b^* c^* \cup a^* bb^* c^* \cup a^* b^* cc^*$
$aa^* b^* c^* \cup bb^* c^* \cup cc^*$
$(aa^* b^* \cup bb^* \cup c)c^*$
$((aa^* \cup b)b^* \cup c)c^*$

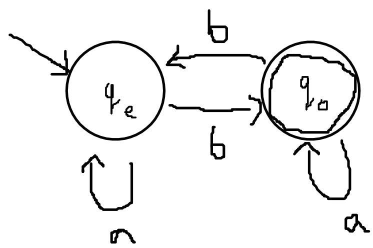
DFA



Example

A DFA to accept all strings over $\{a, b\}$ with an odd number of b's
$Q = \{q_{\text{even}}, q_{\text{odd}}\}$
$\Sigma = \{a, b\}$
$F = \{q_{\text{odd}}\}$
$q_0 = q_{\text{even}}$

δ	a	b
q_{even}	q_{even}	q_{odd}
q_{odd}	q_{odd}	q_{even}



Example

A DFA for $(a \cup b)^*(aa \cup bb)(a \cup b)^*$

a→ (last symbol a) a→
→ () b ↓↑ a → ((aa or bb seen)) ← a, b
b→ (last symbol b) b→

Proof of Theorem

$M_1 = (Q_1, \Sigma, \delta_1, q_{0,1}, F_1)$
$M_2 = (Q_2, \Sigma, \delta_2, q_{0,2}, F_2)$

The new machine simulates both M_1 and M_2 simultaneously.

Create $M = (Q, \Sigma, \delta, q_0, F)$
$Q = Q_1 \times Q_2 = \{[p, q] : p \in Q_1, q \in Q_2\}$
$\delta : Q \times \Sigma \to Q$
$\delta([p, q], a) = [\delta_1(p, a), \delta_2(q, a)]$
$q_0 = [q_{0,1}, q_{0,2}]$
$F = F_1 \times F_2$

Now have to prove that the construction works. $(L(M) = L(M_1) \cap L(M_2))$
Use induction on $|x|$ to prove that
$\delta(q_0, x) = [\delta_1(q_{0,1}, x), \delta_2(q_{0,2}, x)] \;\; \forall x \in \Sigma^*$
Then prove strings accepted are the same

For union, change F to $(Q_1 \times F_2) \cup (F_1 \times Q_2)$

Extensions to DFAs

January-15-13 10:04 AM

NFA Model Definition

An NFA is $M = (Q, \Sigma, \delta, q_0, F)$
All same except δ:
$\delta: Q \times \Sigma \to \mathcal{P}(Q)$

$\mathcal{P}(Q)$ is the power set of Q (also written 2^Q)

Language Accepted

$L(M) = \{x \in \Sigma^* : \delta(q_0, x) \cap F \neq \emptyset\}$

with the extended transition function:
$\delta(r, \epsilon) = \{r\}, \quad r \in Q$

$$\delta(r, xa) = \bigcup_{p \in \delta(r,x)} \delta(p, a), \quad x \in \Sigma^*, a \in \Sigma$$

Theorem

Given an NFA $M = (Q, \Sigma, \delta, q_0, F)$
$\exists$ a DFA $M' = (Q', \Sigma, \delta', q_0', F')$
such that $L(M) = L(M')$

Epsilon Transitions

Allow machine to go from state p to state q without eating up any input.
Called NFA-ϵ or ϵ-NFA

$\delta: Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$
Transition function of NFA-ϵ

Nondeterminism

Rabin & Scott (1959)

- Generalized transition diagram.
  Each node has $|\Sigma|$ arrows out.

  With nondeterminism allow different numbers of arrows out. Allow repeats and allow missing arrows.
- Acceptance corresponds to the existence of an accepting path on input
- Can think of the machine as having processes or threads that are spawned on duplicate transitions

- Can also think of the automaton as 'guessing and checking'. Makes guesses then checks in parallel
  - Example: $L = \{$all strings over $\{a, c, t\}$ with the subword cat in them$\}$
    Guess the start of 'cat' and head off on the chain:
    $\to (q_0) \to \text{c} \to (q_1) \to \text{a} \to (q_2) \to \text{t} \to ((q_3))$
    a, c, t a, c, t

    As a tree for input *cacat*
    c a c a t
    $q_0 \to q_0 \to q_0 \to q_0 \to q_0 \to q_0$
    $\to q_1 \to q_2 \to ((q_3))$
    $q_1 \to q_2 \to \text{x}$

Size Reduction

NFAs can provably save states compared to DFAs
$L_n = \{x \in \{0,1\}^* : x$ has a 1 n positions from the end$\}$

L_4 is accepted by the NFA
$\to (\square) \to 1 \to (\square) \to 0,1 \to (\square) \to 0,1 \to (\square) \to 0,1 \to ((\square))$
0, 1

L_n can be accepted by an NFA with $n+1$ states. Every DFA for L_n needs $\geq 2^n$ states

Proof of Theorem

Let
$Q' = \mathcal{P}(Q)$

$$\delta'(p, a) = \bigcup_{r \in p} \delta(r, a)$$

$q_0' = \{q_0\}$
$F' = \{ q \in Q' : q \cap F \neq \emptyset\}$

Claim

$L(M') = L(M)$

$L(M) = \{x \in \Sigma^* : \ \delta(q_0, x) \cap F \neq \emptyset\}$
$L(M') = \{x \in \Sigma^* : \delta'(q_0', x) \in F'\} = \{x \in \Sigma^* : \delta'(q_0', x) \cap F \neq \emptyset\}$

This suggests showing $\delta(q_0, x) = \delta'(q_0', x)$ by induction on $|x|$
Base case: $|x| = 0, \ x = \epsilon$
$\delta'(q_0', \epsilon) = q_0' = \{q_0\} = \delta(q_0, \epsilon)$
Assume true for $|x| = n$; prove for $|x| = n+1$
$x = ya, \quad a \in \Sigma$
$\delta'(q_0', x) = \delta'(q_0', ya) = \delta'(\delta'(q_0', y), a) = \delta'(\delta(q_0, y), a)$ by induction

$$= \bigcup_{p \in \delta(q_0, y)} \delta(p, a) = \delta(q_0, x)$$

■

Simulating NFA-ϵ with NFA

Can break the NFA-ϵ into sections of a single letter followed by 0 or more ϵ-transitions.

$\mathrm{E}(q) = \{ p :$ you can reach p from q following only 0 or more ϵ-labeled transitions$\}$
$\mathrm{E}(R) = \{ p :$ you can reach p from some state of R following only 0 or more ϵ-labeled transitions$\}$
$R \subseteq Q$

NFA-ϵ $M = (Q, \Sigma, \delta, q_0, F)$
Simulate with NFA
$M' = (Q', \Sigma, \delta', q_0', F')$
$\delta'(p, a) = \mathrm{E}(\delta(p, a))$

But still have ϵ-transitions on start state.

generalized NFA (gNFA)

Allow starting **set** of states instead of a single start state.
$M = (Q, \Sigma, \delta, S, F), \quad S \subseteq Q$
$S = \mathrm{E}(\{q_0\})$

gNFA can by simulated by a DFA the same way as an NFA.

Kleene's Theorem

January-17-13 10:17 AM

Simulation Power

NFA-ε
↓
gNFA NFA
↓ ↓
DFA

↓ - can be simulated by

Theorem (Kleene)

The class of languages accepted by

- gNFA
- NFA-ε
- DFA

is the same as the class of languages specified by regular expressions.

Proved by State Elimination algorithm and Theorem 1.

Theorem 1

Given a regular expression we can construct an NFA-ε for it, with:

- 1 initial state,
- 1 final state,
- no transitions into the initial state, and
- no transitions leaving the final state.

Furthermore, the NFA-ϵ has at most $2n+2$ states and $4n+1$ transitions if the regular expression has n operators.

Generalized NFA (GNFA)

Like a NFA, gNFA, NFA-ε except that transitions can be labeled by arbitrary regular expressions.

Claim

GNFA can be simulated by NFA-ε.
Proof: Replace regular expression edges with the NFA-ε generated like in the proof of Theorem 1.

Proof of Theorem 1

Given r we can write it as one of

- $a, \quad a \in \Sigma$
- ϵ
- $\emptyset$
- $(r_1)(r_2)$
- $r_1 \cup r_2$
- r_1^*

Proof by induction on # of operators in r
Base case: 0 operators

$a \in \Sigma$	→ () → a → (())
ϵ	→ () → ε → (())
$\emptyset$	→ () (())

Concatenation:

$(r_1)(r_2)$
r_1: → [() → ... → (())]
r_2: → [() → ... → (())]

Convert to
→ [() → ... → ()] → ϵ → [() → ... → (())]

Union:

$r_1 \cup r_2$
r_1: → [() → ... → (())]
r_2: → [() → ... → (())]

Convert to:
ϵ / → [() → ... → ()] → \ ϵ
→ () - → (())
ϵ \ → [() → ... → ()] → / ϵ

Kleene Star

r_1^*
→ () → ϵ → [() → ... → ()] → ϵ → (())
\ \ ← ϵ ←/ /
\ /
\ ————— → ϵ → ——/

State Elimination

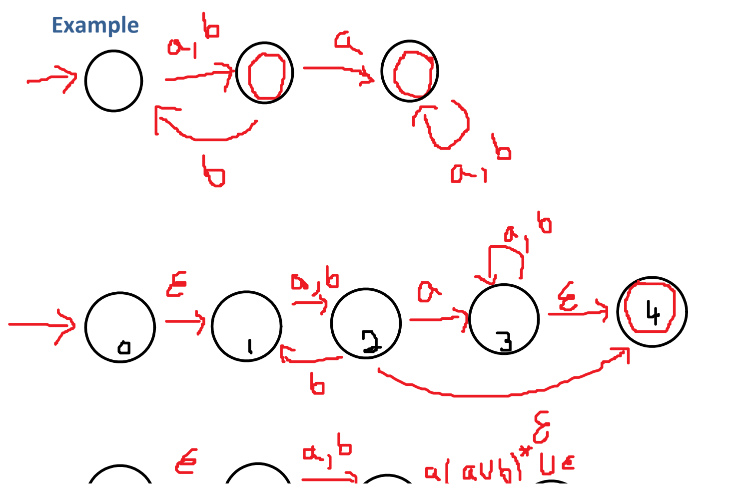
1. Convert your gNFA to one with
   - 1 initial state
   - 1 final state
   - no transitions into initial state
   - no transitions leaving final state

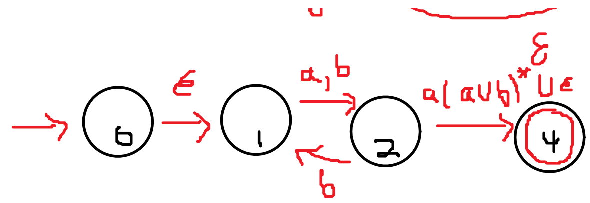
   by adding extra states and ε-transitions
2. Pick a state P to eliminate that is neither the initial nor the final state.
   P has:
   - incoming transitions $r_1, \dots, r_n$
   - outgoing transitions $s_1, \dots, s_m$
   - transition to self t

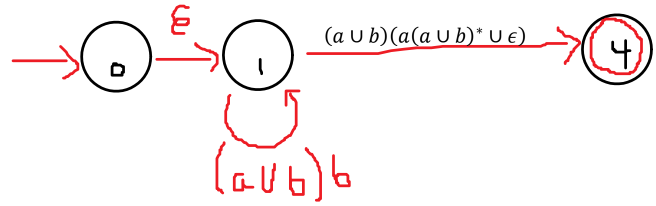
   For input state A and output state B corresponding to r_i and s_j add transition from A to B labelled by $(r_i)(t)^*(s_j)$
   Combine transitions from states using union.
3. Keep going until only initial and final state left

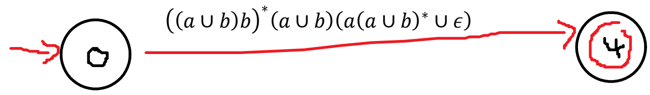
Rough strategy: try to pick states with few input/output transitions for elimination.

Example




ε
a, b
b
a(a∪b)* ∪ ε
0
1
2
4


ε
(a ∪ b)(a(a ∪ b)* ∪ ϵ)
(a ∪ b)b
0
1
4


((a ∪ b)b)*(a ∪ b)(a(a ∪ b)* ∪ ϵ)
0
4

Closure Properties

January-17-13 11:01 AM

Theorem

The class REG is closed under the operations:

1. Union - Yes
2. Concatenation - Yes
3. Star - Yes
4. Intersection - Yes
5. Complement - Yes
   - Take a DFA for L and flip the "finality" of each state.

Closure Properties

We say a class of languages $\mathcal{C}$ is closed under a (unary, binary) operator $\otimes$ if

- $\forall\, L \in \mathcal{C}, \quad \otimes L \in \mathcal{C}$, or
- $\forall L_1, L_2 \in \mathcal{C}, \quad L_1 \otimes L_2 \in \mathcal{C}$

Prefix

$\text{Pref}(L) = \{x \in \Sigma^* : \exists y \in L \text{ s.t. } x \text{ is a prefix of y}\}, \qquad L \subseteq \Sigma^*$

Example

$L = \{\text{cat}, \text{dog}\}$

$\text{Pref}(L) = \{\epsilon, \text{c}, \text{ca}, \text{cat}, \text{d}, \text{do}, \text{dog}\}$

Theorem

If L is a regular language then so is $\text{Pref}(L)$

If L is regular, is L^R regular?

Theorem

Yes

Proof of Theorem ($\text{Pref}(L)$)

Take DFA $M = (Q, \Sigma, \delta, q_0, F)$

Change F to $\{q \in Q : \exists y \in \Sigma^* \;\; s.t. \delta(q, y) \in F\}$

Call new machine M'

Claim

$L(M') = \text{Pref}(L(M))$

$x \in L(M') \Leftrightarrow \delta(q_0, x) \in F'$

$\Leftrightarrow \delta(q_0, x) \in \{q \in Q : \exists y \in \Sigma^* : \delta(q, y) \in F\}$

$\Leftrightarrow \exists y\; \delta(\delta(q_0, x), y) \in F$

$\Leftrightarrow \exists y\; \delta(q_0, xy) \in F$

$\Leftrightarrow \exists y\; xy \in L(M)$

$\Leftrightarrow x \in \text{Pref}(L(M))$

Proof of Theorem (Reversal)

Given a DFA $M = (Q, \Sigma, \delta, q_0, F)$

Create a gNFA $M' = (Q, \Sigma, \delta', S, F')$

$S = F$

$F' = \{q_0\}$

$\delta'(p, q) = \{q \in Q : \delta(q, a) = p\}$

Non - Regular Languages

January-22-13 10:00 AM

Pumping Lemma for Regular Languages

(Iteration Lemma)

If L is regular then
$\exists$ a constant $n \geq 1$ (which could depend on L)
$\forall z \in L, \quad |z| \geq n$
$\exists u, v, w$ such that $z = uvw,\ |uv| \leq n,\ |v| \geq 1$
$\forall i \geq 0\ uv^i w \in L$

Contrapositive

If $\forall$ constants $n \geq 1$
$\exists\, z \in L, \quad |z| \geq n$
$\forall u, v, w$ such that $z = uvw,\ |uv| \leq n,\ |v| \geq 1$
$\exists i$ s.t. $uv^i w \notin L$
then L is not regular

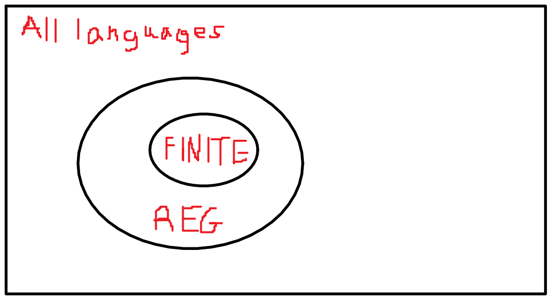
Using this in proofs

Think of game vs. adversary
$\forall$ - an adversary
$\exists$ - your choice

Warning

See also "Nine errors students commonly make when using the pumping lemma" on course website

Hierarchy of Languages



All Languages - Uncountable
REG, FINITE - Countable

Proofs of Non-Regular Languages Using Pumping Lemma

Example 1

$L = \{0^n 1^n : n \geq 0\} = \{\epsilon, 01, 0011, 000111, \ldots\}$

Adversary: n
You pick: $z = 0^n 1^n$
Adversary: $z = uvw,\ |uv| \leq n,\ |v| \geq 1$
$\quad u = 0^a, \quad v = 0^b, \quad w = 0^{n-(a+b)} 1^n$
$\quad a + b \leq n, \quad b \geq 1, \quad a \geq 0$
You pick: $i = 0,\ uv^i w = uw = 0^{n-b} 1^n \notin L$

Example 2

$\text{PRIMES}_1 = \{\text{the prime numbers in unary}\} = \{a^2, a^3, a^5, a^7, \ldots\}$
$= \{aa, aaa, aaaaa, \ldots\}$

Adversary: n
Pick $z \in \text{PRIMES}_1,\ |z| \geq n$
$\quad$ By Euclid $\exists$ prime $p > n$ Pick $z = a^p$
Adversary: $z = uvw,\ |uv| \leq n,\ |v| \geq 1$
$\quad u = a^j, \quad v = a^k, \quad w = a^{p-(j+k)}$
$\quad j + k \leq n, \quad k \geq 1$
Pick i such that $uv^i w \notin \text{PRIMES}_1$
$\quad |uv^i w| = j + ik + p - (j + k) = p + (i - 1)k$
$\quad$ Pick $i = p + 1$
$\quad uv^i w = a^{p(k+1)} \notin \text{PRIMES}_1$
$\quad p(k + 1)$ is a multiple of p but not equal to p since $k \geq 1$ and $k + 1 \geq 2$

Proof of the Pumping Lemma

L is regular so $\exists$ a DFA M accepting it.
Let $n = \#$ of states in M

If $|z|$ has length t with no repeating states then the DFA visits $t + 1$ states.
Therefore, if $|z| \geq n$ there is a repeated state.

$$\to (q_0) \xrightarrow{u} \ldots \to (q) \xrightarrow{w} \ldots \to ((q_f))$$
$$/\ \backslash \quad \xleftarrow{} \quad v$$

Since $\delta(q_0, \epsilon), \delta(q_0, z[1]), \ldots, \delta(q_0, z)$ has $\geq n + 1$ states, some state is repeated. Call the first repeated state q.
Let v be the first path from q back to q. Then $uv^i w$ still labels an accepting path and so $uv^i w \in L$
$|v| \geq 1$ from our definition of the path (1st time hitting q to $2nd$)
$|uv| \leq n$ Consider the length-n prefix of z; it most involve $n + 1$ states and so some state is repeated.

More Non-Regular Languages

Example 3

$\text{PRIMES}_2 = \{\text{the prime numbers in binary}\} = \{10, 11, 101, 111, \ldots\}$
Adversary: n
You pick $z \in \text{PRIMES}_2,\ [z] > 2^n$
$\quad$ Notation: $[z] =$ Integer represented by z in base 2
Adversary: $z = uvw,\ |uv| \leq n,\ |v| \geq 1$
Pick: Hope to show $uv^i w \notin \text{PRIMES}_2$
$\qquad$ i.e. $[uv^i w]$ is not a prime number

$[uvw] = [u] \cdot 2^{|vw|} + [v] \cdot 2^{|w|} + [w]$

$$[uv^iw] = [u]\cdot 2^{|v^iw|} + \sum_{j=1}^{i}[v]\cdot 2^{|v^{i-2}w|} + [w]$$
$$= [u]\cdot 2^{i|v|+|w|} + [v]\cdot 2^{|w|}\cdot\left(2^{(i-1)|v|} + 2^{(i-2)|v|} + \cdots + 2^{|v|} + 1\right) + [w]$$
$$= [u]\cdot 2^{i|v|+|w|} + [v]\cdot 2^{|w|}\cdot \frac{2^{i|v|}-1}{2^{|v|}-1} + [w] \quad \text{(recall } |v| \neq 0 \; so \; 2^{|v|}-1 \neq 0\text{)}$$

Recall

Fermat's theorem says
$a^p \equiv a \; (mod\ p)$ if p is prime
So maybe pick $i = p$?

Compute
$[uv^pw] - [uvw] \; (mod\ p)$

$[u]\cdot 2^{p|v|+|w|} \equiv [u]\cdot 2^{|v|+|w|} \equiv [u]\cdot 2^{|vw|}$

If $2^{|v|} \equiv 1 \; (mod\ p)$ then $\left(2^{(p-1)|v|} + \cdots + 2^{|v|} + 1\right) \equiv 0 \; (mod\ p)$
otherwise
$$[v]\cdot 2^{|w|}\cdot \frac{2^{p|v|}-1}{2^{|v|}-1} \equiv [v]\cdot 2^{|w|}\cdot \frac{2^{|v|}-1}{2^{|v|}-1} = [v]\cdot 2^{|w|}$$

$\therefore [uv^pw] - [uvw] \equiv 0 \; (mod\ p)$
And hence $[uv^pw]$ is divisible by p and $[uv^pw] > p$ ■

Decision Problems

January-22-13 11:00 AM

Theorem 1
If an n-state DFA accepts any string x it accepts an x such that $|x| < n$

Theorem 2
L is infinite iff M accepts some $x, n \leq |x| < 2n$

Questions about regular languages
1. Is $L = \emptyset$?
2. Is L infinite?
3. Given L_1, L_2 is $L_1 = L_2$

Difficulty could depend upon representation.
Represent L with:
 DFA, NFA (clear box model)
 DFA (black box model)
 - Don't know mechanics of DFA, only a DFA that accepts or rejects strings.
 - Know Σ
 - Know n, upper bound on the number of states
 All you can ask is: given x does M accept x

Clear Box Model
1) Is $L \neq \emptyset$?
 Use graph search algorithm (BFS, DFS, etc.) to look for a path from q_0 to a final state.
 DFA: n states $O(n)$
 NFA: n states, m transitions $O(n+m)$
2) Is L infinite?
 Look for a cycle, reachable from q_0 and from which you can reach a final state
 Search for reachability from q_0, search for reachability from final states, search for cycle with DFS.
 DFA $O(n)$
 NFA $O(n+m)$
3) Given L_1, L_2 is $L_1 = L_2$?
 DFA - One way is to compute unique minimal DFA's in $O(n \log n)$

 $L_1 = L_2$ iff $(L_1 - L_2) \cup (L_2 - L_1) = \emptyset$
 make a DFA accepting $(L_1 - L_2) \cup (L_2 - L_1)$. Has $O(n^2)$ states so
 DFA $O(n^2)$
 NFA PSPACE-complete

Proof of Theorem 1
Let x be the shortest string accepted.
If $|x| \geq n$ then there is a loop in the states of the DFA that can be cut out to get a shorter string.

Proof of Theorem 2
L infinite but M accepts no string x with $n \leq |x| < 2n$
M must accept some string of length $\geq n$ so let z be the shortest such string. If $n \leq z < 2n$, contradiction. So $|z| \geq 2n$.
By pumping lemma, $z = uvw$, $|uv| \leq n$, $|v| \geq 1$ and $uv^0w \in L$
$|uw| < |z|$
But $|uw| = |z| - |v| \geq 2n - n = n$
So uw is a shorter string in L with $|uw| \geq n$, contradiction.

Now suppose M accepts x, $n \leq |x| < 2n$
By proof of pumping lemma $\exists u, v, w$ such that $x = uvw$ $|v| \geq 1$
$uv^iw \in L \ \ \forall i \geq 1$

Black Box Model
Ask questions like, is $x \in L(M)$?
Know Σ and n, the number of states of M
1) Is $L \neq \emptyset$
 Try all strings up to length n
2) Is L infinite?
 Try all strings x such that $n \leq |x| < 2n$
3) Is $L_1 = L_2$?
 Last time we constructed a DFA for $L = (L_1 - L_2) \cup (L_2 - L_1)$
 $L = \emptyset$ iff $L_1 = L_2$
 by our construction (similar to the intersection one) there exists a DFA of n^2 states accepting L.
 Can simulate L by checking if any string is accepted by L_1 or L_2 and not the other.
 Algorithm: test each string of length 0 to $n^2 - 1$. If any is accepted then $L_1 \neq L_2$ otherwise $L_1 = L_2$

Context-Free Languages

January-24-13 10:25 AM

Notation

V - A finite set of variables (typically A, B, C, ...)
Σ - A finite set of "terminals"
S - start variable $S \in V$
P - finite set of "productions" or "rules" that tell how strings are derived
$G = (V, \Sigma, P, S)$

Context-free grammars have productions like $A \to \alpha_1 \alpha_2 \dots \alpha_n,\ n \geq 0$
where $A \in V$
$\alpha_1, \alpha_2, \dots, \alpha_n \in (V \cup \Sigma)^*$

$A \to \alpha,\ A \to \beta,$ write in the form $A \to \alpha | \beta$

Derivations

If $A \to \alpha$ is a production, $\alpha \in (V \cup \Sigma)^*$ and $\beta, \gamma \in (V \cup \Sigma)^*$
then $\beta A \gamma \Rightarrow \beta \alpha \gamma$ (one step derivation)
$\Rightarrow$ read as "goes to" or "derives"

We say $\alpha \Rightarrow^* \beta$ if $\exists b_0 = \alpha, b_1, \dots, b_n = \beta$ such that
$\beta_0 \Rightarrow \beta_1, \beta_1 \Rightarrow \beta_2, \dots, \beta_{n-1} \Rightarrow \beta_n$

$\alpha \Rightarrow^n \beta$ means α derives β in n steps.

Note

Derivations are not necessarily unique

Leftmost and Rightmost Derivation

Leftmost derivation: replace leftmost variable at each step
Rightmost derivation: replace rightmost variable at each step

Language

G a grammar, $G = (V, \Sigma, P, S)$
Then $L(G) = \{x \in \Sigma^* : S \Rightarrow^* x\}$

Ambiguous Grammar

A grammar G is ambiguous if

1) $\exists\, w \in L(G)$ such that w has 2 different parse trees.
2) $\exists\, w \in L(G)$ such that w has 2 different leftmost derivations.
3) $\exists\, w \in L(G)$ such that w has 2 different rightmost derivations.

Sentential Form

Any string of variables and terminals, particularly intermediate step of a derivation.

History

Panini c. 400 BC

Sanskrit grammar, 3959 rules

Chomsky 1956 - 1959

Equivalence between grammar specifications and machines

Backus and Naur, late 1950's

Grammars for programming languages

Example Non-unique Derivations

$S \to AB$
$A \to a$
$B \to b$

$S \Rightarrow AB \Rightarrow aB \Rightarrow ab$
$S \Rightarrow AB \Rightarrow Ab \Rightarrow ab$

Parse Tree

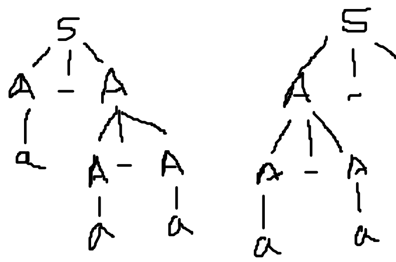
S at root
children of a node $A, A \in V$ ware the symbols of rhs in left-to-right order.

S
A B
a b

To remove ambiguity in derivation order use leftmost or rightmost derivation. Left with only true ambiguity.

Example of Ambiguity

$S \to A - A$
$A \to a \mid A - A$



$a - a - a$ evaluates to a $a - a - a$ evaluates to $-a$

Simple Grammar Example: Palindromes

$\text{PAL} = \{x \in \{a, b\}^* : x = x^R\}$
$S \to \epsilon | a | b | aSa | bSb$

$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abbSbba \Rightarrow abbabba$

Goal: $L(G) = \text{PAL}$

1. $L(G) \subseteq \text{PAL}$
   Let x be derived in G, show it is a palindrome
   Typically: do an induction on the length of the derivation.
   - Sometimes need hypotheses on all strings derived from S, not just terminal strings.

2. $\text{PAL} \subseteq L(G)$
   Given $x \in \text{PAL}$, construct a derivation for it.
   Typically: do an induction on $|x|$

Example

$G = S \to aSb | ab$
$L = \{a^i b^i : i \geq 1\}$

1. $L(G) \subseteq L$
   If $S \Rightarrow^i x \in \{a, b\}^*$ then $x = a^i b^i$

   Proof by induction on i
   Base case: $i = 1$
   $S \Rightarrow ab = a^1 b^1 \in L$

   Assume true for $i - 1$, prove for i.
   $S \Rightarrow^i x$
   $S \Rightarrow^1 aSb \Rightarrow^{i-1} x$
   So $x = ayb$ and $S \Rightarrow^{i-1} y$
   By induction, $y = a^{i-1} b^{i-1}$ so $x = a a^{i-1} b^{i-1} b = a^i b^i \in L$

2. $L \subseteq L(G)$
   Take $x \in L$, write $x = a^i b^i$. Show by induction on i that $x \in L(G)$

Base case: $i = 1$ $S \Rightarrow ab$
Otherwise assume true for $i-1$, prove for i
∃ a derivation $S \Rightarrow^* a^{i-1}b^{i-1}$ by induction
$S \Rightarrow aSb \Rightarrow^* aa^{i-1}b^{i-1}b = a^i b^i$

Example

$S \to aB|a$
$B \to bS|b$
$L = a(ba)^* \cup ab(ab)^*$
Want to prove $L(G) = L$

Strategy: include both S and B in induction hypothesis

Example

$S \to AASB|AAB$
$A \to a$
$B \to bbb$

Strategy: make induction hypothesis that covers all sentential forms (any string derived from S)
This is called an **invariant.**

CFG Example

January-29-13 10:06 AM

$L(G) \subseteq L$
$L \subseteq L(G)$

G:
$S \to a$
$S \to aS$
$S \to bSS$
$S \to SSb$
$S \to SbS$

$L = \{x \in \{a,b\}^* : |x|_a > |x|_b\}$

Claim:
If $|w|_a > |w|_b$ then $\exists$ a derivation $S \Rightarrow^* w$

Proof: By induction on $|w|$

Base case: $|w| = 1,\ w = a,$ derivation $S \Rightarrow a$
No can claim true for $|w| \leq n$ Prove it for $|w| = n + 1$
Case 1:
$w = by, \quad |y| = n$
$|w|_a - |w|_b \geq 1$ base case $w \in L$
$|y|_a - |y|_b \geq 2$ suggests trying to write $y = y_1 y_2$
$D(t) = |t|_a - |t_b|$

Goal is to write $y = y_1 y_2$ with $D(y_1) \geq 1, D(y_2) \geq 1$
If we can do this then by induction:
$S \Rightarrow^* y_1$
$S \Rightarrow^* y_2$
$S \Rightarrow bSS \Rightarrow^* by_1S \Rightarrow^* by_1y_2 = w$

$D(\epsilon) = 0, \quad D(y) = 2$
For ever additional letter read in the prefix of y, D changes by ± 1 so $\exists$ and prefix y_1 of y such that $D(y_1) = 1,\ D(y_2) \geq 1$
By induction, we have $S \Rightarrow^* y_1, S \Rightarrow^* y_2$
Case 2:
$w = yb$. Mirror image of Case 1
Case 3: w begins and ends with a
3a: $w \in a^+,$ use $S \to aS$ n times followed by $S \to a$ $S \Rightarrow a^{n+1} = w$
3b: w has at least one b
$w = a \ldots . bab \ldots . ba \ldots a$
$w = x_i b z_i$ here the b is the i^{th} b in w
Goal: find i such that $D(x_i) \geq 1,\ D(z_i) \geq 1, \ \ 1 \leq i \leq r$

$D(z_r) \geq 1$
If $D(x_r) \geq 1$ then we're done.
$y_1 = x_r, \quad y_z = z_r, \quad S \Rightarrow SbS$
So assume $D(x_r) \leq 0$
Let m be the smallest index such that $D(x_m) \leq 0$
$D(x_1) \geq 1$ so $m \geq 2$
That means x_{m-1} is a string in this list. So $D(x_{m-1}) \geq 1$
Is $D(z_{m-1}) \geq 1$?

x_{m-1}	b	z_{m-1}	z_{m-1}
x_m	x_m	b	z_m

$D(x_m) \geq D(x_{m-1}) - 1$
$D(x_m) \leq 0$
$0 \geq D(x_m) \geq D(x_{n-1}) - 1$
$\Rightarrow D(x_{m-1}) \leq 1 \Rightarrow D(x_{m-1}) = 1$
$\Rightarrow D(z_{m-1}) \geq 1$
By induction
$S \Rightarrow^* x_{m-1}$
$S \Rightarrow^* z_{m-1}$
$S \Rightarrow^* SbS$
$S \Rightarrow^* x_{m-1} b z_{m-1} = w$

Chomsky Normal Form

January-29-13 10:40 AM

Chomsky Normal Form (CNF)

Every production is of the form
$A \to BC, \quad B, C \in V$
$A \to a, \quad a \in \Sigma$
If G is in CNF then $\epsilon \notin L(G)$

Theorem

If L is in CFL, $\epsilon \notin L$ then $\exists$ a CNF for L grammar.

Algorithm for CNF

1. Get rid of useless variables
   A variable $A \in V$ is **useless** if it does not appear in a sentential form in any derivation of the form $S \Rightarrow^* w$
   Ways of being useless:
   - The variable might not produce any terminal strings
   - The variable might not appear in any derivation starting from S

   a) Run DTS on G
      Throw away all variables & production involving variables that are in $V - DTS(V)$
      V' =new set, G' new grammar
   b) Run RV on G' throw away all variables & production involving variables that are in $V' - RV(V')$
2. Replace all terminals that appear in a RHS with length ≥ 2
   Introduce new variable that goes directly to the variable.
   $A_a \to a$
3. Shorten RHS in large productions
   Add new variables to break up large production
4. Remove ϵ-productions: $A \to \epsilon$
   a) Identify all variables A such that $A \Rightarrow^* \epsilon$ then replace A by ϵ in the RHS of every production involving A.
5. Remove unit productions
   A unit production is $A \Rightarrow^* B, \;\; A, B \in V$
   We find all productions of the form $B \to \alpha, \;\; \alpha \in V^*$ and add the production $A \to \alpha$ provided $|\alpha| \neq 1$
   Must do this for every pair of variables. Could square the size of the grammar.

`DTS(G) /* Variables that derive terminal strings */`
$T \coloneqq \{A : \exists$ a production $A \to w, \quad w \in \Sigma^*\}$
$T' \coloneqq \emptyset$
`while` $(T \neq T')$ `do`
$T' \coloneqq T$
$T \coloneqq T \cup \{A : \exists$ a production $A \to \alpha, \quad \alpha \in (\Sigma \cup T)^*\}$
`return T`

`RV(G) /* Variables reachable from S */`
$T \coloneqq \{S\}$
$T' \coloneqq \emptyset$
`while` $(T \neq T')$ `do`
$T' \coloneqq T$
$T \coloneqq T \cup \{A : \exists B \in T \;\; s.t. \; B \to \alpha A \beta\}$
`return(T)`

Example of DGS

$S \to AB|CF$
$A \to a$
$B \to CD$
$C \to c$
$D \to d$
$E \to FEF$

$T = \{A, C, D\}, \quad T' = \emptyset$
$T' = \{A, C, D\}, \quad T = \{A, C, D, B\}$
$T' = \{A, C, D, B\}, \quad T = \{A, C, D, B, S\}$
$T' = \{A, C, D, B, S\}, \quad T = \{A, C, D, B, S\}$

Example Production Shortening

$A \to BCDEF$
Replace that with
$A \to BA_1, \quad A_1 \to CA_2, \quad A_2 \to DA_3, \quad A_3 \to EF$
With new variables A_1, A_2, A_3

Example Removing ϵ-productions

$S \to AB$
$A \to C$
$C \to \epsilon$
$B \to b$
$A \to DA$

$A \Rightarrow^* \epsilon, A \Rightarrow^* \epsilon$
Add the productions:
$S \to B, \quad A \to D$
And remove
$C \to \epsilon$

Pushdown Automaton & Closure Properties

January-31-13 10:04 AM

Context-free languages (CFL's)
- Language generated by CFG's

CNF - Chomsky normal form
- Handout on it on home page

Closure Properties of CFL's
- Union
- Concatenation
- Star
- Complement? No
- Intersection? No

Theorem
L_1, L_2 are CFL's then so are
$L_1 \cup L_2$
$L_1 L_2$
L_1^*

Pushdown Automaton (PDA)
Consists of
Q : Finite set of states
Σ : Input alphabet
Γ : Stack alphabet
δ : Transition function
$q_0 \in Q$: Initial state
$F \subseteq Q$: Set of final states

$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to 2^{Q \times (\Gamma \cup \{\epsilon\})}$
$\delta(p, q, X) = (q, Y)$
  p state
  a input symbol
  X popped stack element
  q new state
  Y pushed stack element

Definition: $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$

Instantaneous Description (ID)
Triple of (state, unexpended input, stack)
(q, Σ^*, Γ^*)
Convention: Top of the stack to left

$ID \vdash ID'$ "goes to"

Language
$L(M) = \{x \in \Sigma^* : (q_0, x, \epsilon) \vdash^* (q, \epsilon, \alpha) \text{ for } q \in F, \alpha \in \Gamma^*\}$

Proof of Theorem
Union
Let
$G_1 = (V_1, \Sigma, P_1, S_1)$ for L_1
$G_3 = (V_2, \Sigma, P_2, S_2)$ for L_2
produce $G = (V, \Sigma, P, S)$ for $L_1 \cup L_2$
$P = P_1 \cup P_2 \cup \{S \to S_1, S \to S_2\}$
$V = V_1 \cup V_2 \cup \{S\}$

Assuming that $V_1 \cap V_2 = \emptyset$ and $S \notin V_1$ and $S \notin V_2$
If not, just rename variables.

Need to see $L(G) = L(G_1) \cup L(G_2)$

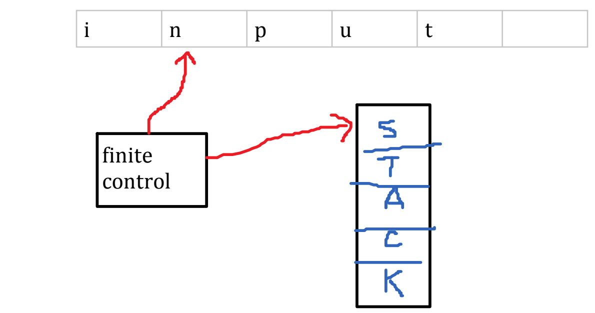
Concatenation
$S \to S_1 S_2$
add this production

Star
$S \to SS_1 | \epsilon$

Pushdown Automaton (PDA)
Finite automaton: finite # of states
Turing machine: potentially unbounded states

Pushdown automata: potentially unbounded storage
- 1 stack

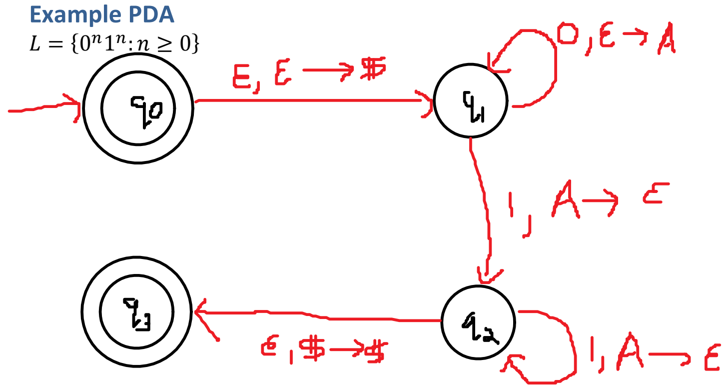
In some sense between finite automata and Turing machines
No writing on tape, reading left to right



PDA is inherently nondeterministic unless otherwise specified.
The deterministic version describes a smaller class of languages.

PDAs can
input tape, either read a single symbol or not (ϵ- move)
stack
- stack oblivious move
  - push
  - leave stack the same
- depends on op of stack
  - replace symbol on top
  - pop

Example PDA
$L = \{0^n 1^n : n \geq 0\}$

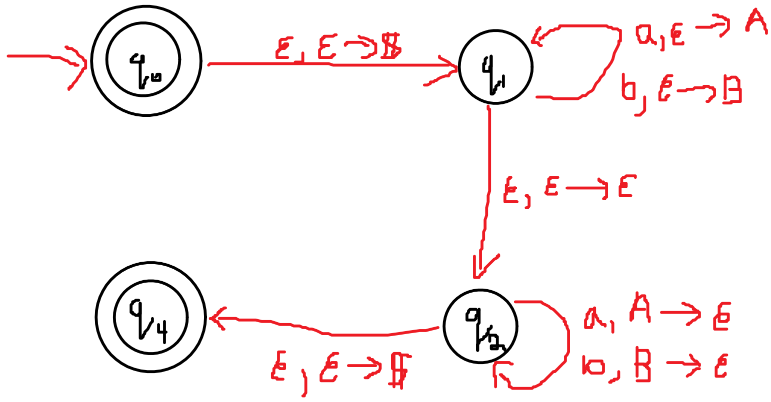


State	Input	Stack Contents
q_0	00111	ε
q_1	00111	\$
q_1	0111	A\$
q_1	111	AA\$
q_2	11	A\$
q_2	1	\$
$((q3))$	1	\$

The string accepted is 0011, not 00111

Example

EVENPAL $= \{ww^* \;: w \in \{a, b\}^*\}$



CFG, PDA Equivalence

February-05-13 10:07 AM

Theorem

Given a CFG $G = (V, \Sigma, P, S)$ there exists a PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ such that $L(G) = L(M)$

Theorem

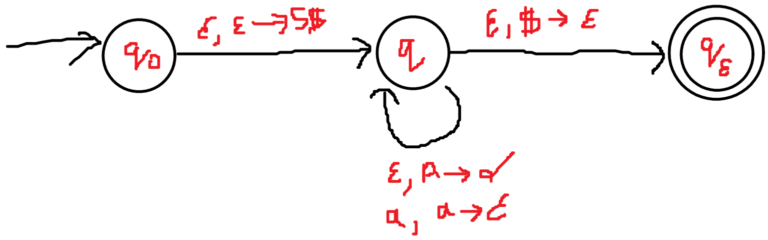
If L is a CFL and R is a regular language then $L \cap R$ is a context free language.

Do cross product of states in L and R and have stack follow L

Proof of Theorem

Idea: Store suffix of sentinel on stack. Match terminal in prefix against input.
Goal: $S \Rightarrow^* x\alpha,\ x \in \Sigma^*, \alpha \in (V \cup \Sigma^*)$ iff $(q, x, \epsilon) \vdash^* (q, \epsilon, \alpha)$

CFG → PDA



$A \in V$
$A \to \alpha \in G$
$a \in \Sigma$

Notation (q, x, α) is (state, unexpended input, stack contents)

Want to prove

$S \Rightarrow^* x\beta, \quad b \in (V \cup \Sigma)^*, \quad x \in \Sigma^*$
iff
$(q, x, S\$) \vdash^* (q, \epsilon, \beta\$)$

Proof

Assume $(q, x, S\$) \vdash^i (q, \epsilon, \beta\$)$
Prove by induction on i that $S \Rightarrow^* x\beta$

Base case: $i = 0$ (need to fill this in)
Induction step: Assume hypothesis is true for all $t < i$ Prove for $t = i$ to get $S \Rightarrow^* x\beta$
$(q, x, S\$) \vdash^{<i} (\ldots) \vdash (q, \epsilon, \beta\$)$

Case 1: There was a variable A on top of the stack and to get β I pushed to rhs of an A-production.
$(q, x, S\$) \vdash^{<i} (q, \epsilon, A\gamma\$) \vdash (q, \epsilon, \beta\$)$
$A \to \alpha$ then $\beta = \alpha\gamma$
By induction, $S \Rightarrow^* xA\gamma \Rightarrow x\alpha\gamma = x\beta$

Case 2: There was a letter input matched against stack
$(q, x, S\$) \vdash^{<i} (q, a, a\beta\$) \vdash (q, \epsilon, \beta\$)$
$x = ya$ for some y
$(q, ya, S\$) \vdash^* (q, a, a\beta\$)$
$\Rightarrow (q, y, S\$) \vdash^* (q, \epsilon, a\beta\$)$
By induction, $S \Rightarrow^* ya\beta = x\beta$

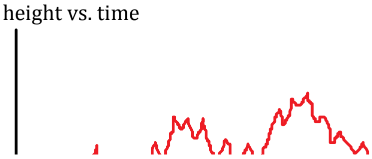
Assume $S \Rightarrow^i x\beta$
Want to conclude that $(q, x, S\$) \vdash^* (q, \epsilon, \beta\$)$
Base case $i = 0$. Check .
Induction:
$S \Rightarrow^* xA\gamma \Rightarrow x\beta,$ so $\beta = \alpha\gamma$
$A \to \alpha$
By induction,
$(q, x, S\$) \vdash^* (q, \epsilon, A\gamma\$)$
so
$(q, x, S\$) \vdash^* (q, \epsilon, A\gamma\$) \vdash^* (q, \epsilon, \alpha\gamma\$) = (q, \epsilon, \beta\$)$

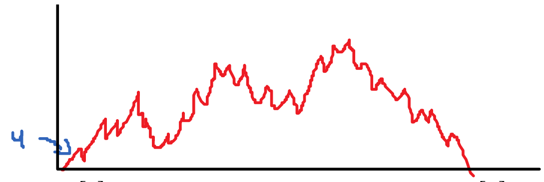
PDA → CFG

$PDA = M(Q, \Sigma, \Gamma, \delta, q_0, F)$
Idea: define grammar symbol $A_{pq},\ (p, q \in Q)$ such that $A_{pq} \Rightarrow^* x$ iff $(p, x, \epsilon) \vdash^* (q, \epsilon, \epsilon)$

Assumptions:
- Only 1 final state q_f (can just make a new one and point all old ones to it)
- The stack is empty when q_f is reached (can add a new state before the final state that removes everything on the stack)
- Every move pushes or pops a stack symbol (can simulate no change by a push followed by a pop)

$p = q, A_{pp} \to \epsilon$
Stack height vs. time





$x[1]$ ————————————————> $x[n]$
p q

This is case 1. In case 2 the stack might return to height 0 between $x[1]$ and $x[n]$
The first push pushes u. In case 2 then the last pop is u

Case 1:
$A_{pq} \to aA_{rs}b$ if
$\delta(p,a,\epsilon) \ni (r,U)$ and $\delta(s,b,U) \ni (q,\epsilon)$
$\forall a,b, \exists \Sigma \cup \{\epsilon\}, \quad U \in \Gamma$
$p,q,r,s \in Q$

Case 2:
$A_{pq} \to A_{pr}A_{rq} \forall r \in Q$

Start state of the grammar $A_{q_0 q_f}$

Proof

$A_{pq} \Rightarrow^* x$ iff $(p,x,\epsilon) \vdash^* (q,\epsilon,\epsilon)$ $(*)$
Prove $(*)$ by induction.

By induction on i, where $A_{pq} \Rightarrow^i x$
Base case $i = 1$
$A_{pq} \Rightarrow x$ so $p = q$ and production $A_{pp} \to \epsilon$ so $(p,\epsilon,\epsilon) \vdash (q,\epsilon,\epsilon)$
Induction step: Assume $\Rightarrow$ of $(*)$ holds for $< i$ step derivations and prove for i.
$A_{pq} \Rightarrow^i x$
Case 1: 1st step of the derivation is $A_{pq} \to aA_{rs}b$
$A_{pq} \Rightarrow aA_{rs}b \Rightarrow^{<i} x$ so $x = ayb$ and $A_{rs} \Rightarrow^{<i} y$ by induction, $(r,y,\epsilon) \vdash^* (s,\epsilon,\epsilon)$
$\Rightarrow (r,yb,\epsilon) \vdash^* (s,b,\epsilon)$
$(p,x,\epsilon) = (p,ayb,\epsilon) \vdash (r,yb,U) \vdash^* (s,b,U) \vdash (q,\epsilon,\epsilon)$
Case 2: 1st step of the derivation is $A_{pq} \to A_{pr}A_{rs}$
$A_{pq} \Rightarrow A_{pr}A_{rq} \Rightarrow^* x, \quad x = yz$
$A_{pr} \Rightarrow^* y,$ by induction $(p,y,\epsilon) \vdash^* (r,\epsilon,\epsilon) \Rightarrow (p,yz,\epsilon) \vdash^* (r,z,\epsilon)$
$A_{rq} \Rightarrow^* z,$ by induction $(r,z,\epsilon) \vdash^* (q,\epsilon,\epsilon)$
$(p,x,\epsilon) = (p,yz,\epsilon) \vdash^* (r,z,\epsilon) \vdash^* (q,\epsilon,\epsilon)$

Assume $(p,x,\epsilon) \vdash^i (q,\epsilon,\epsilon)$ $(+)$
Want to show $A_{pq} \Rightarrow^* x$ by induction on i.
Base case $i = 0$: Then $p = q$. $\exists$ a derivation $A_{pp} \Rightarrow^* \epsilon$
Induction: Assume $(+)$ is true for all $t < i$; prove for i.
$(p,x,\epsilon) \vdash^i (q,\epsilon,\epsilon), \ x = ayb$
Case 1: stack height always > 0 until end.
Case 2: stack height hits 0 at some intermediate point of computation.

Case 1:
$(p,x,\epsilon) = (p,ayb,\epsilon) \vdash (r,yb,U) \vdash^* (s,b,U) \vdash (q,\epsilon,\epsilon)$
$(p,ayb,\epsilon) \vdash (r,yb,U) \Rightarrow (r,U) \in \delta(p,a,\epsilon)$
$(s,b,U) \vdash (q,\epsilon,\epsilon) \Rightarrow (q,\epsilon) \in \delta(s,b,U)$
so in grammar $\exists$ a production $A_{pq} \to aA_{rs}b$

Also have $(r,y,U) \vdash^* (s,\epsilon,U)$ and $(r,y,\epsilon) \vdash^* (s,\epsilon,\epsilon)$
$A_{pq} \Rightarrow aA_{rs}b \Rightarrow^* ayb = x$ (by induction)
Case 2:
$(p,x,\epsilon) \vdash^* (r,z,\epsilon) \vdash^* (q,\epsilon,\epsilon)$
$x = yz$
By induction, $(r,z,\epsilon) \vdash^* (q,\epsilon,\epsilon)$ means $A_{rq} \Rightarrow^* z$ is in the grammar
$(p,yz,\epsilon) \vdash^* (r,z,\epsilon) \Rightarrow (p,y,\epsilon) \vdash^* (p,\epsilon,\epsilon)$ so by induction $A_{pr} \Rightarrow^* y$
$A_{pq} \Rightarrow A_{pr}A_{rq} \Rightarrow^* yA_{rq} \Rightarrow yz = x$

Pumping Lemma for CFLs

February-07-13 10:01 AM

Pumping Lemma for CFL's

If L is a CFL then
$\exists n \geq 1$
$\forall z \in L, \quad |z| \geq n$
$\exists z = uvwxy, \quad |vwx| \leq n, \quad |vx| \geq 1$
$\forall i \geq 0, \quad uv^iwx^iy \in L$

Contrapositive of Pumping Lemma for CFL's

If $\forall n \geq 1$
$\exists z \in L, |z| \geq n$
$\forall z = uvwxy, \quad |vwx| \leq n, \quad |vx| \geq 1$
$\exists i \geq 0$ such that $uv^iwx^iy \notin L$
then L is not a CFL

Theorem (Intersection)

If L_1, L_2 are CFL's then $L_1 \cap L_2$ need not be a CFL.

Theorem (Complement)

If L is a CFL, then $\bar{L}$ need not be.

Open Problem

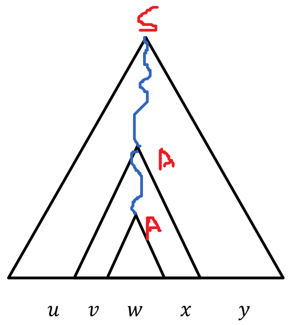
Let $\Sigma = \{0, 1\}$
Let P be the language of powers $P = \{x^n : x \notin \epsilon, n \geq 2\} = \{00, 11, 000, 111, 0000, 0101, 1010, 1111, \dots\}$
P is not a CFL

Consider $\bar{P}$, the set of non-powers ("primitive words")

$\bar{P} = \{\epsilon\} \cup \{0, 1, 01, 10, 001, \dots\}$
Is $\bar{P}$ context free?

Quick "Proof" of Pumping Lemma

A long $z \Rightarrow$ a long path in a parse tree for z
(from root to a leaf)
A long path $\Rightarrow$ some variable is repeated along path



So
1. $S \Rightarrow^* uAy$
2. $A \Rightarrow^* vAx$
3. $A \Rightarrow^* w$

Do 1. then 2. i times then 3. to get
$uv^iwx^iy \in L$

Application

$L = \{a^nb^nc^n : n \geq 0\}$
Len n be chosen.
Pick $z \in L, |z| \geq n$
$z = a^nb^nc^n$
$z = uvwxy \ |uvx| \leq n \ |vx| \geq 1$

Case 1: If either v or x contains two different kinds of letters, then by pumping with $i = 2$
$uv^2wx^2y = uvvwxxy \notin L$
Case 2: Now b and x each separately contain one type of letter.
2a) v, x both contain the same letter (repeated some number of times), not both empty
pump with $i = 0$: $vwy \notin L$ because that letter will have $< n$ copies in the resulting string but the others still have n.
2b) v contains one kind of letter, x contains another
pump with $i = 0$. The third letter stays at n but some other has $< n$ copies

Proof of Theorem (Intersection)

By counterexample
Take
$L_1 = \{a^nb^nc^r : n, r \geq 0\} = \{a^nb^n : n \geq 0\}c^*$
$L_2 = \{a^rb^nc^n : n, r \geq 0\} = a^*\{b^nc^n : n \geq 0\}$
So
$L_1 \cap L_2 = \{a^nb^nc^n : n \geq 0\}$

L_1 and L_2 are CFL's and $L_1 \cap L_2$ is not

Proof of Theorem (Complement)

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ so closed under complement & union $\Rightarrow$ closed under intersection.
Know closed under union and not under intersection, so not closed under complement.

Alternate counterexample proof:
Proved on assignment 3 that $L = \overline{\{ww : w \in \Sigma^*\}}$ is a CFL
but $\bar{L} = \{ww : w \in \Sigma^*\}$ is not a CFL by pumping lemma.

Application

$\{ss : s \in \{0, 1\}^*\}$ is not a CFL
Bad choice:
$z = 1^{2n}$
$z = uvwxy, \quad |vwx| \leq n, \quad |vx| \geq 1$
Suppose $v = 11, \quad x = \epsilon$
Then we're in trouble and no contradiction possible

Better choice
$z = 0^n1^n0^n1^n \in L$
Case 1: vwx lies in the first half of z
Use $i = 0$. The middle will shift so the 1st half of the new string ends in 0's
2nd half of the new string ends in 1's
Case 2: vwx lies entirely in 2nd half
Same thing as case 1
Case 3: vwx straddles the boundary between the first and second halves.
Pump with $i = 0$
Either the 0s in the first half will be less than the 0s in the second half, or same for the 1's

Proof of the Pumping Lemma

L context-free $\Rightarrow \exists$ a CNF grammar G for $L - \{\epsilon\}$
Construct a parse tree for z in G

Lemma

Let T be a parse tree for a string z in a grammar in CNF.
If all paths from the root to the leaf are of length $\leq t$ then $|z| = 2^{t-1}$
Length of a path is the number of edges.

Proof of lemma by induction on t
Base case: $t = 1$

$S \Rightarrow a$, path of length 1. $z = a$, $|z| = 2^0$

Induction assume true for t and prove for $t + 1$
$|z| \leq 2^{t-1} + 2^{t-1} = 2^t$ ■

So $|z| > 2^{t-1} \Rightarrow \exists$ a path of length $> t$
Let G have r variables.
$|z| > 2^r \Rightarrow \exists$a path of length $\geq r + 2$

Each edge comes from a variable so $r + 2$ variables, so some variable is repeated.
take $n = 2^r + 1$ where $r = \#$ of variables.

Some variable is repeated along some path. Consider the 2nd occurrence of the first repeated variable, A, going up from the bottom.
$S \Rightarrow^* uAy$
$A \Rightarrow^* vAx$
$A \Rightarrow^* w$
Path from bottom to 2nd A is $\leq r + 1$ so $|vwz| \leq 2^r \leq n$ by lemma
The first occurrence of A lies in exactly one subtree of the second subtree of A. The other subtree of A must generate some terminals so $|vx| \geq 1$

Turing Machines

February-07-13 11:12 AM

Formal Turing Machine

A TM is
$M = (Q, \Sigma, \Gamma, \delta, q_0, p_r)$
Q: finite set of states
Σ: finite nonempty input alphabet ($\sqcup \notin \Sigma$)
Γ: finite tape alphabet, $\Sigma \subseteq \Gamma$, $\sqcup \in \Gamma$
q_a: accept state $\in Q$
q_r: reject state $\in Q$
$\delta: (Q - \{q_a, q_r\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$

Configuration

A configuration of a TM is a string from $\Gamma^* Q \Gamma^*$ of the form xqy, $x \in \Gamma^*, q \in Q, y \in \Gamma^*$
It means
current state is q
current tape contents (up to last non-black) is xy
current symbol being scanned is first symbol of y

Goes To

$\vdash$ "goes to"
$\vdash^*$ "goes to after 0 or more one moves"
Relates configurations as one would expect.

Accepting / Recognizing

$L(M) =$ language accepted / recognized by a TM
$= \{x \in \Sigma^* : q_0 x \vdash^* y q_a z \text{ for some } y, z \in \Gamma^*\}$

Behaviours

A TM has 3 behaviours
1) it eventually reaches q_a - accept
2) it eventually reaches q_f - reject
3) "loops"

Decision

L is **decided** by M if $L = L(M)$ and further, M halts on every input (either reach q_a or q_r)

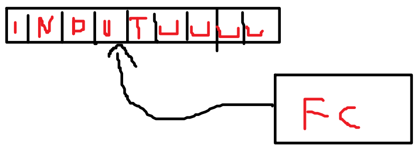
Recursively Enumerable (RE)

A language is called recursively enumerable if it is accepted by a Turing machine.

A language is called recursive if it is decided by a Turing machine.

Turing Machine

- Finite control
- An unbounded tape
  - Holds a finite input
  - In basic model the tape has a left edge
- Can both read and write on the tape
- Transitions:
  Based on the current state and contents of the cell being scanned
  - move to a new state, rewrite current cell contents, move left or right



Transition function
$\delta(q, p) = (q', a, R)$

Paper on the subject (Turing 1936)
Had different definition of "computer". Meant a person doing computation.

Sipser's Model

two distinguished states: $q_{\text{accept}} \to q_a, q_f$
$q_{\text{reject}} \to q_r$
No transitions out of these states

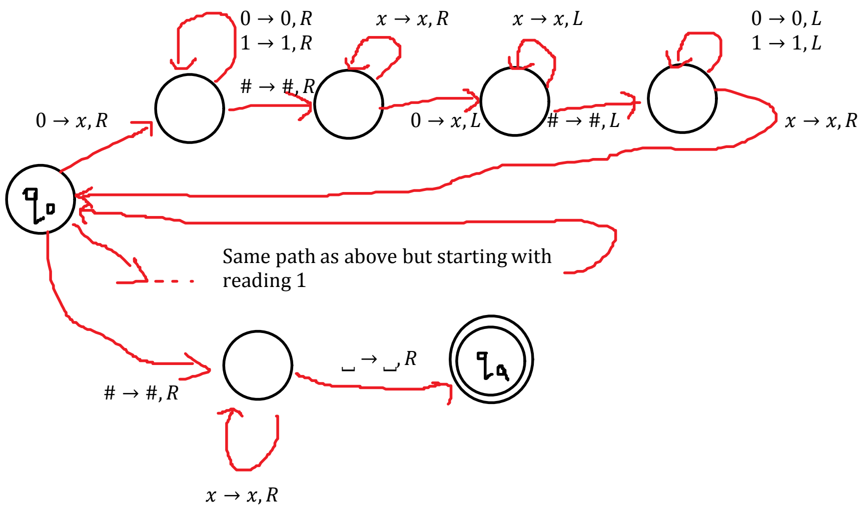
Turing machines must move right (R) or left (L) on each move.
A move left at cell 0 stays in cell 0.
There is always a move (based on current state & current symbol scanned) except from q_a and q_r
After input there are arbitrarily many blank "$\sqcup$"

Example

$\{w\#w : w \in \{0,1\}^*\} \subseteq \Sigma^*$, $\Sigma = \{0, 1, \#\}$

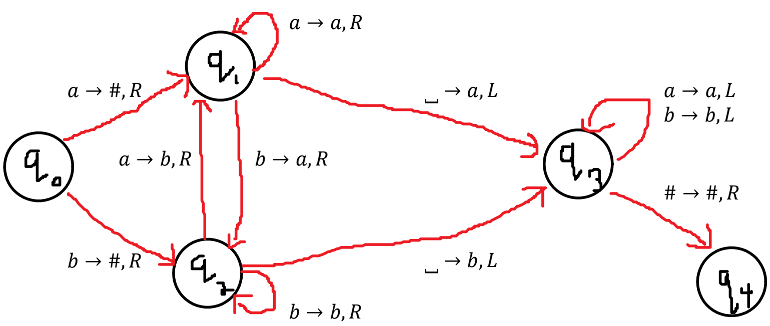
Example

0	1	0	#	0	1	0	$\sqcup$	$\sqcup$	$\sqcup$



Subroutine

Move input down 1 cell and insert a delimiter at the front.
Assume input is over $\{a, b\}$



Example of Configurations

$x, y \in \Gamma^*, \qquad a, b, c \in \Gamma, \qquad p \in Q$

Using $\delta(p, q) = (q, b, L)$:

$\qquad xcpay \vdash xqcby$

Using $\delta(p, q) = (q, b, R)$

$\qquad xpay \vdash xbqy$

Language Hierarchy

All languages | R.E. | Recursive | CFL | REG | FINITE

Variations on Turing Machines

February-14-13 10:07 AM

Variations

1. Allow S moves (stationary)
   $\boxed{q} \rightarrow (a \rightarrow b, S) \rightarrow \boxed{r}$
   Simulated by
   $\boxed{q} \rightarrow (a \rightarrow b, R) \rightarrow \boxed{q'} \rightarrow \begin{pmatrix} \forall c \in \Gamma \\ c \rightarrow c, L \end{pmatrix} \rightarrow \boxed{r}$

2. Tape has "tracks" (but 1 tape head)

a	b	a	␣
a	a	b	␣

   $\delta(p, [a, b]) = (q, [a', b'], R)$
   2 - track example:
   $\Gamma = \Sigma \cup \{\_\} \cup (\Sigma \cup \{\_\}) \times (\Sigma \cup \{\_\})$
   In general can allow arbitrarily many tracks by manipulating tape alphabet to allow tuples.

   Does not require changing the TM at all.

3. Multiple tapes with independent heads
   New transitions function:
   $\delta(q, a_1, a_2, \ldots, a_t) = (p, a_1', a_2', \ldots, a_t', d_1, d_2, \ldots, d_t)$
   d_i is direction for i^{th} head.

4. Two way infinite tape

   Use two tracks. Top track stores right hand side, bottom track stores flipped left hand side, with special marker symbol on cell 1 on first cell.

   `␣␣␣␣input␣␣␣␣`
   Simulated by

   `put␣␣␣␣␣`
   `Δni␣␣␣␣␣`

5. Nondeterminism
   Instead of transition function
   $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$
   the nondeterministic Turing machine has
   $\delta: Q \times \Gamma \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$

Example Multitape Turing Machine

Use a multitape TM to accept $L = \{a^{k^2} : k \geq 1\}$

Idea: Use a 3-tape Turing machine
tape 1: input
tape 2: hold J X's ($J = 1, 2, 3, \ldots$)
tape 3: hold J^2 X's

Steps:
1. Write X on tape 2 & 3 and return head to the left
2. If tapes (1) and (3) contain the same number of symbols, halt (accept)
3. If tape (3) contains more X's then tape (1) contains a's then reject
4. Copy the context of tape (2) to the end of tape (3) twice, then add one more X to tapes (2) and (3)
   $J \rightarrow J + 1$
   $J^2 \rightarrow J^2 + 2J + 1 = (J + 1)^2$
5. Go to step 2

Example

Tapes
`Δaaaaaaaaa`
`ΔX`
`ΔX`
↓
`Δaaaaaaaaa`
`ΔXX`
`ΔXXXX`
↓
`Δaaaaaaaaa`
`ΔXXX`
`ΔXXXXXXXXX`
Accept

Simulating a Multitape TM

Suppose M is a k-tape TM
Let M' be a TM with a tape with $(2K + 1)$ tracks

Track 1: Hold # in cell 1
Track 2, 4, 6, ...: hold contents on tape 1, 2, 3, ...
Track 3, 5, 7, ...: hold the head indicators of table 1, 2, 3, ...
to simulate M:
   For every odd track i
   - move right on track 3 to find ↑ (head marker)
   - store corresponding symbol in $i - 1$ in finite control
   - then return to # in track 1
   - Repeat for the next odd track.

   When all symbols have been accumulated, perform symbol rewriting on each track then move ↑ accordingly on each track.

Example Nondeterministic Turing Machine

Using a nondeterministic TM accept $L = \{1^k : k \text{ is composite}\}$ (composite meaning non-prime ≥ 4)
Idea: Use a 4-tape nondeterministic TM
Tape 1: input
Tape 2: $\_1^i : i \geq 2$
Tape 3: $\_1^j : j \geq 2$
Tape 4: to compute $1^{i \times j}$

Steps:
1. Write ␣11 on tape 2, then choose nondeterministically between: writing more 1's and advancing to step 2.
2. Write ␣11 on tape 3, then choose nondeterministically between: writing more 1's and advancing to step 3.
3. Copy tape 2 to tape 4 and advance tape head on tape 3.
4. Repeat 3 until tape head on tape 3 scans blank.
5. When done, compare tape 4 to tape 1. If equal, accept.

Theorem

Every nondeterministic TM M can be simulated by a deterministic TM M'

Idea

The computation of M can be represented as a tree where each node splits when a nondeterministic choice is made.

Claim: Branching factor is finite.
$\delta \rightarrow 2^{Q \times \Gamma \times \{L,R\}}$ so can branch at most $|Q \times L \times \{L, R\}| = b$ times at each node.

Can use numbers in base b to denote branches of computation of M. (By indexing all the edges from each node from 0 to $\leq b - 1$)

M' deterministic TM with 3 tapes
Tape 1: Holds input
Tape 2: We use it exactly as machine M uses its tape
Tape 3: Successively holds numbers in base b

Ex: $(b = 2)$ $\{0, 1, 10, 11, 100, 101, \ldots\}$

M' does the following:

1. Copy tape 1 into tape 2
2. Simulate M on tape 2

    When must make a non-deterministic choice (including when just one option), use number at appropriate position in tape 3 to make the decision. Stop when made all decisions represented by Tape 3 or when an invalid decision is represented.

3. If accept state of M is reach, halt and accept.
    Otherwise update tape 3, erase tape 2, and repeat from 1.

This traverses the tree in BFS order so it will terminate eventually if any of the possible paths are accepting.

Enumerators & RAM

February-14-13 11:03 AM

Enumerator

A Turing machine with a write-only output tape.
Starts only with tape blank.
Prints out strings of L, in any order, maybe with duplicate.
010, 1010, 10, … an order of strings of L

Theorem

A languages is Turing recognizable iff there exists an enumerator E for it.

Random Access Machine Model (RAM)

Features

- Read-only input tape
- Write-only output tape
- random access memory
- finite program
- each tape square and memory cell can hold arbitrarily large integer
- accumulator - register 0 - where arithmetic can be performed

Theorem

If L is enumerated by E, on input X, run E and search output tape for X. If it appears, accept.

Suppose L is Turing recognizable. Let $S_1, S_2, S_3, \ldots$ be an ordering of Σ^*, then do the following

1. For $i = 1, 2, 3, \ldots$ do
    2. For j= 1 to i do
        3. If M accepts s_j within i steps, write s_j on output tape.

Turing Machine Simulation of RAM

See hopefully posted TM tapes for RAM
Simulation:

1. set program counter on tape 6 to 1
2. repeat until HALT detected
    a. fetch instruction from tape 1
    b. execute instruction
    c. update program counter

Universal Turing Machine

February-28-13 10:05 AM

The universal Turing machine T_U

- can simulate every Turing machine TM
- input to TU is a Turing machine T (encoded as a string)
  - and encoding of an input w to T
  - T_U runs T on w and does exactly what T would do on input w:
    - accept
    - reject
    - loop for ever

Theorem

There exists a universal TM T_U with input alphabet $\{0, 1\}$ that on input $\langle T, w \rangle$ will simulate T on w and do what T does on input w.

Corollary

The language

$\{\langle T, w \rangle : T \text{ accepts } w\}$

is Turing-recognizable (recursively enumerable)

(Can check for input that does not represent a TM as we simulate)

Encoding a Turing Machine

1. List all alphabet symbols (assume the set of all possible symbols is countable)
   a. Label them s_i
2. Encode letter s_i as $0 \ldots 0 = 0^i$
3. Encode string $z = a_1 a_2 \ldots a_n$ as $110^{b_1}10^{b_2} \ldots 10^{b_n}1$ b_i is the code for letter a_i
4. Encode moves of TM

   $\delta(p, a) = (q, b, D), D \in \{L, R\}$

   $1e(p)1e(a)1e(q)1e(b)1e(d)$

   $q_0 = 0, q_1 = 00, q_2 = 000, q_3 = 0000, \ldots$

   $R = 0, L = 00$
5. Encode TM by encoding each element of its transition function

   $e(q_a)1e(q_r)1e(t_1)1e(t_2)1 \ldots 1e(t_i)111$

   (Encode accept and reject state at the beginning)

Details unimportant

1. Uniquely decodable
2. Tell when it ends (prefix-free encoding)

Proof of Theorem

T_U has a tape holding an encoding of T's input tape.

T_U needs:

1. A tape to hold T's input tape in encoded form
2. A tape to hold $\langle T \rangle$
3. A work tape, current state

Repeat until T reaches q_a or q_r

Fetch state from tape 3

Look on tape 2 for matching instruction

Carry it out

Update new state

Decision Problems

February-28-13 10:55 AM

Decision Problem

A question with a parameter and a yes/no answer.

Can encode a decision problem as a language.
$\{\langle X\rangle : X$ is a "yes" instance of that decision problem$\}$

Acceptance

Two types of TM acceptance

Equivalent:

- allow non-halting if $x \notin L(M)$
- language is Turing-recognizable
- recursively enumerable

Equivalent:

- must always halt
- language is Turing decidable
- recursive

We say a decision problem is solvable or decidable if ∃ an always-halting TM deciding the language associated with the problem.

Accepts(w) Decision Problem

Given a TM T and an input w does T accept w?
Does there exist a TM A to solve this problem?
No.

We have proved that
$A_{TM} \coloneqq \{\langle T, w\rangle : \text{TM } T \text{ accepts } w\}$
is recursively enumerable but not recursive.

Halting Problem

Given a TM T and an input w does T halt on w?
Unsolvable.

General Technique

For proving unsolvability.
Assume it is solvable. Us a TM that solves it as a subroutine to solve a known unsolvable problem.

Accepts(ε) Problem

(The blank tape problem)
Given a TM T, does T accept ϵ?
This is unsolvable.

Is Empty Problem

Given T, is $L(T) = \emptyset$?
Unsolvable.

Accepts(REG)

Given a TM T, is $L(T)$ a regular language?
This is unsolvable.

Example Decision Problems

Is n a prime?
Does $\langle G\rangle$ have a Hamiltonian cycle?

Example Language

Primes $\{10, 11, 101, \dots, \}$

Hilbert's 10th Problem

Diophantine equation: polynomial equation with integer coefficients.
Do there exist integer values of variables making it true?

Russell's Paradox

$T = \{S : S \notin S\}$
Is $T \in T$?
Both $T \in T$ and $T \notin T$ lead to a contradiction
Therefore T cannot exist.

Solving

What does it mean to solve a decision problem?

- There exists a TM that always halts, either accepting or rejecting to solve the problem
- L is recursive (Turing -decidable)

We want a method that

1) Is **complete**: always answers "yes" or "no"
2) Is **correct**: always gives the correct answer
3) Is **objective** of **mechanistic**: no judgment involved, every step is clear
4) Is **finitely describable**
5) Is **deterministic**
6) Always eventually answers

Barber Problem

Barber B cuts the hair of everyone (and only those) in Kitchener who does not cut their own hair.

Accepts(w) Problem

Assume A exists.
Modify A as follows:

B: _ accept if T accepts $\langle T\rangle$
$\rightarrow \langle T\rangle \rightarrow \boxed{\rightarrow \langle T, \langle T\rangle\rangle \rightarrow \boxed{A} <}$ __reject if T rejects $\langle T\rangle$

Now flip output of A

C: — accept if T rejects $\langle T\rangle$
$\rightarrow \langle T\rangle \rightarrow \boxed{\rightarrow \langle T, \langle T\rangle\rangle \rightarrow \boxed{A} >\!<}$ _ reject if T accepts $\langle T\rangle$

C:

- takes $\langle T\rangle$ as input
- halts and rejects if T accepts $\langle T\rangle$
- halts and accepts if T doesn't accept $\langle T\rangle$

Now run C on input $\langle C\rangle$
Contradiction: A does not exist.

Halting Problem

Assume such an H exists

accept if T halts on w
$\langle T, w\rangle \rightarrow \boxed{H} <$ reject otherwise

Want to make

accept if T accepts w
$\langle T, w\rangle \rightarrow \boxed{A} <$ reject if T does not accept w

Use A:

____Accept
$\boxed{\langle T, w\rangle \rightarrow \boxed{H} < \begin{smallmatrix} w \rightarrow \boxed{T} < \\ \text{-----} \ \text{-} \end{smallmatrix}} >$ — Reject

A solves Accept(w)
Contradiction, so H does not exist.

Accepts(ε) Problem

Assume a TM BT exists which does
$\langle T\rangle \rightarrow \boxed{BT}$ accept if T accepts ϵ, reject if T rejects ϵ
Construct

accept if T accepts w
$\langle T, w\rangle \rightarrow \boxed{\langle T'\rangle \rightarrow \boxed{BT} <}$ reject if T does not accept w

Want T accepts w iff T' accepts ϵ
How T' behaves:

it erases its tape
writes w to its tape
simulates T

Is Empty Problem

Assume TM E exists which solves this problem

$\langle T \rangle \to \boxed{E} <$ accept if $L(T) = \emptyset$
reject if $L(T) \neq \emptyset$

Use E to construct

$\langle T \rangle \to \boxed{\langle T' \rangle \to \boxed{E} < \begin{matrix} L(T') = \emptyset - \\ L(T') \neq \emptyset - \end{matrix}}$ $-$reject if T does not accept ϵ
$-$accept if T accepts ϵ

T': Looks at its input. If input is ϵ, simulates T.
If input is not ϵ it rejects.

Accepts(REG) Problem

Assume it is solvable

$\langle T \rangle \to \boxed{AR} <$ accept if $L(T)$ is regular
reject if $L(T)$ is not regular

Construct

$\langle T, w \rangle \to \boxed{\langle T' \rangle \to \boxed{AR} <}$ accept if T accepts w
reject if T rejects w

Want T' such that $L(T')$ is regular iff T accepts w

Idea:

$$L(T') = \begin{cases} \{0^n 1^n\} & \text{if } T \text{ does not accept } w \\ \Sigma^* & \text{if } T \text{ does accept } w \end{cases}$$

Make T' as follows:
T' examines its input. If it is $0^n 1^n$ for some $n \geq 0$, it accepts & halts.
Otherwise, simulate T on w.

Rice's Theorem

March-07-13 10:05 AM

Theorem

If L and $\bar{L}$ are both Turing-recognizable (r.e.) then L and $\bar{L}$ are both Turing-decidable (recursive)

Corollary

If L is Turing-recognizable but not Turing-decidable then $\bar{L}$ is not Turing-recognizable.

Property of a Language

Collection of languages having that property
$P = \{\{a^*\}, \{a^*b\}, \{(a+b)^*a\}, \dots\}$

Nontrivial Property

At least on r.e. language has the property and at least one does not.
Nontrivial means $P \neq \emptyset$ and $P \neq$ All RE Languages

Rice's Theorem

If P is a nontrivial property then the decision problem
Given M, does $L(M)$ have the property P?
is unsolvable.

Proof of Theorem

There exists a TM M_1 accepting L_1 and M_2 accepting L_2
We create a TM that on input x

- simulates M_1 on input x on tape 1
- simulates M_2 on input x on tape 2

alternating steps (1st M_1, then M_2, etc.)
wait until either M_1 or M_2 halts and accepts

- if it's M_1 halt and accept
- if it's M_2 halt and reject

Proof of Corollary

Suppose $\bar{L}$ were Turing-recognizable. By the Theorem, L is Turing-decidable a contradiction.

Example

$A_{TM} = \{\langle M, w\rangle : M \text{ accepts } w\}$
A_{TM} is Turing-recognizable (a universal TM accepts it)
A_{TM} is not Turing-decidable
so $\bar{A}_{TM}$ is not Turing-recognizable

$A = \{\langle M, w\rangle : M \text{ does not accept } w\}$
$\bar{A}_{TM} = A \cup \{x : x \neq \langle M, w\rangle\ \forall M, w\}$
But $\{x : x \neq \langle M, w\rangle \forall M, w\}$, the set of invalid encodings, is Turing-decidable.
So A is not Turing-recognizable.

Proof

Assume it is solvable.

$\langle M\rangle \rightarrow \boxed{P\text{-solver}} <$ accept if $L(M)$ has property P / reject if $L(M)$ has property P

Assume $\emptyset \notin P$ (If that is not the case, think of $\bar{P}$)
Then let A be any TM such that $L(A)$ does have the property P

$\langle M, w\rangle \rightarrow \boxed{\langle T\rangle \rightarrow \boxed{P\text{-solver}} < \begin{matrix} L(M) \in P \\ L(M) \notin P \end{matrix}}\begin{matrix} \text{—accept if } w \in L(M) \\ \text{—reject if } w \notin L(M) \end{matrix}$

We create T to do the following
On input x, T simulates M on w

- if M halts & rejects, T rejects
- if M and accepts, T runs A on x
  - do whatever A does

$$L(T) = \begin{cases} L(A) & \text{if } M \text{ accepts } w \\ \emptyset & \text{if } M \text{ does not accept } w \end{cases}$$

Post Correspondence Problem

March-07-13 10:41 AM

Post Correspondence Problem

Emil Post

You have dominoes $\boxed{\dfrac{abc}{da}}$

- Have a string on top and on bottom. Cannot be flipped
- Finite number of distinct types
- as many as you want of each type

A match in PCP is a list of dominoes where there concatenation of the upper entries exactly equals the concatenation of the lower entries.

PCP problem:
given a list, is there a match?

PCP Decision problem

Given tile list, is there a match?
$L_{PCP} = \{\langle L\rangle : L \text{ has a match}\}$
Typical encoding
0#1?1#011?001#0??

Modified PCP

Just like PCP but get to specify which tile goes first.

Example PCP

$\boxed{\dfrac{001}{00}}, \boxed{\dfrac{11}{011}}, \boxed{\dfrac{01}{000}}, \boxed{\dfrac{010}{10}}$

Match attempt:
$\boxed{\dfrac{001}{00}}\boxed{\dfrac{010}{10}}\boxed{\dfrac{101}{10}}\ldots$

No match possible. The first domino must be the one show, leaving one more 1 in the top than the bottom, and no other domino can increase the number of 1's in the bottom relative to the top.

Example PCP

$\boxed{\dfrac{01}{0}}, \boxed{\dfrac{1}{11}}$

Yes, there is a match

Example PCP

$\boxed{\dfrac{0}{1}}, \boxed{\dfrac{1}{011}}, \boxed{\dfrac{011}{0}}$

Yes, there is a match, but shortest has 75 tiles.

PCP Undecidability Proof Sketch

Use the upper and lower entries to record possible TM configurations throughout the course of an accepting computation.

Recall: TM configuration xqw
tape is $xw\sqcup\sqcup\ldots$
state is q
scanning first symbol of w

An accepting computation can be expressed
$q_0w\#\ldots\#\ldots\#\ldots\#xq_hw'\#$

Given $M\&w$ we build dominoes so $\exists$ a match iff M accepts w

- Lower entries will be one computational step ahead of the upper ones.
- If halting state is reach, upper entries are allowed to catch up.

MPCP Undecidability Setup

For input M, w first state is
$\boxed{\dfrac{\#}{\#q_0w_1w_2\ldots w_n\#}}$
where $w = w_1w_2\ldots w_n$

If $\delta(q, a) = (p, b, R)$ add domino
$\boxed{\dfrac{qa}{bp}}$

If $\delta(q, a) = (p, b, L)$ add domino
$\boxed{\dfrac{cqa}{pcb}}, \quad \forall c \in \Gamma$

Also add
$\boxed{\dfrac{a}{a}}, \forall a \in \Gamma, \quad \boxed{\dfrac{\#}{\#}}, \quad \boxed{\dfrac{\#}{\sqcup\#}}$

$\boxed{\dfrac{bq_a}{q_a}}, \quad q_a$ accepting state, $\quad \forall b \in \Gamma$

$\boxed{\dfrac{q_ab}{a_a}}, \quad \forall b \in \Gamma$

$\boxed{\dfrac{q_a\#\#}{\#}}$

Example

$M \to \boxed{q_0} \to (0 \to 1, R) \to \boxed{q_1} \to (1 \to 2, L) \to \boxed{\boxed{q_a}}$

$w = 01$

$\boxed{\dfrac{\#}{\#q_001\#}}\boxed{\dfrac{q_00}{1q_1}}\boxed{\dfrac{1}{1}}\boxed{\dfrac{\#}{\#}}\boxed{\dfrac{1q_11}{q_a12}}\boxed{\dfrac{\#}{\#}}\boxed{\dfrac{q_a1}{q_a}}\boxed{\dfrac{2}{2}}\boxed{\dfrac{\#}{\#}}\boxed{\dfrac{q_a2}{q_a}}\boxed{\dfrac{\#}{\#}}$

MPCP to PCP

Notation
$*u =* u_1 * u_2 * u_3 \ldots * u_n$
$u *= u_1 * u_2 * u_3 * \cdots u_n *$
$*u *=* u_1 * u_2 * u_3 * \cdots * u_n *$

For dominoes

$\boxed{\begin{matrix} t_1 \\ - \\ b_1 \end{matrix}}, \boxed{\begin{matrix} t_2 \\ - \\ b_2 \end{matrix}}, \dots, \boxed{\begin{matrix} t_n \\ - \\ b_n \end{matrix}}$

where must start with $\boxed{\begin{matrix} t_1 \\ - \\ b_1 \end{matrix}}$

replace with dominoes

$\boxed{\begin{matrix} * t_1 \\ - \\ * b_1 * \end{matrix}}, \boxed{\begin{matrix} * t_2 \\ - \\ b_2 * \end{matrix}}, \dots, \boxed{\begin{matrix} * t_n \\ - \\ b_n \end{matrix}}$

Reductions

March-12-13 10:02 AM

Problem Reduction
We say a problem P_1 reduces to a problem P_2 if, given a TM that solves P_2 we can use it as a subroutine to solve P_1.

Write $P_1 \leq_T P_2$

Language Reduction
L_1 reduces to L_2 means "given a TM solving membership in L_2, we can use it to solve membership in L_1"

Write $L_1 \leq_T L_2$

Theorem
If $L_1 \leq_T L_2$ and L_2 is Turing-decidable (recursive) then so is L_1

Theorem (Contrapositive)
If $L_1 \leq_T L_2$ and L_1 is not recursive (not Turing-decidable) then L_2 is not.

Function Computation
We say a TM M computes a function $f(x)$ if
$q_0 x \vdash^* h_a f(x), \qquad f: \Sigma^* \to \Gamma^*$

Many-One Reduction (Mapping Reduction)
We say $L_1 \leq_m L_2$ (L_1 mapping reduces to L_2) iff $\exists$ a computable function f such that $x \in L_1 \Leftrightarrow f(x) \in L_2$

decide membership in L_1

$x \to \boxed{\xrightarrow{f} \boxed{f} \xrightarrow{f(x)} \boxed{\text{membership in } L_2} < \begin{matrix} f(x) \in L_2 - \\ f(x) \notin L_2 - \end{matrix}} \begin{matrix} h_a & x \in L_1 \\ h_r & x \notin L_1 \end{matrix}$

Problems with CFG's

Decision Problem (INT2CFG)
Given two CFG's G_1 and G_2 does there exist $x \in L(G_1) \cap L(G_2)$?
(i.e. does $L(G_1) \cap L(G_2) \neq \emptyset$)

Claim
PCP $\leq_m$ INT2CFG

Ambiguity Problem (AMBIG)
Given a CFG G, is it ambiguous?
(That is, is there a string $x \in L(G)$ with 2 different parse trees?)

PCP $\leq_m$ AMBIG

Other Unsolvable Problems
Tiling problem: Can you tile a quarter-plane with tiles that are coloured on the 4 sides. Adjacent tiles must match colours.
Method, each row can simulate the state of a TM on some input. Can tile only if TM does not halt.

Example Reduction
Element distinctness reduces to sorting.

Proof of Theorem
L_2 recursive implies

$x \to \boxed{M_2 = \text{Membership in } L_2} < \begin{matrix} h_a & \text{if } x \in L_2 \\ h_r & \text{if } x \notin L_2 \end{matrix}$

$L_1 \leq_T L_2$ implies

$z \to \boxed{\sim \to \boxed{M_2} < \begin{matrix} \sim \\ \sim \end{matrix}} \begin{matrix} h_a & z \in L_1 \\ h_r & z \notin L_1 \end{matrix}$

So L_1 is decidable.

Example
$Accepts(w) = \{\langle M, w\rangle : M \text{ accepts } w\}$
$Accepts(\epsilon) = \{\langle M\rangle : M \text{ accepts } w\}$
We showed $Accepts(w) \leq_T Accepts(\epsilon)$

$Accepts(w) \leq MPCP \leq PCP$

Example Mapping Reduction
$Accepts(w) \leq_m Accepts(\epsilon)$
$\langle M, w\rangle \xrightarrow{f} \langle M'\rangle$

$Accepts(w) \leq_m \leq MPCP \leq_m PCP$

Hilbert's 10th Problem
H10A
Given a k-variate polynomial p with coefficients in $\mathbb{Z}$ decide if $\exists$ a k-tuple $\in \mathbb{Z}^k$ for which $p(\dots) = 0$

H10B
Given a k-variate polynomial q with coefficients in $\mathbb{Z}$ decide if $\exists$ a k-tuple $\in \mathbb{N}^k$ for which $q(\dots) = 0$. $\mathbb{N} = \{0,1,2,\dots\}$

H10A $\leq_T$ H10B
Given $p(x,y,z)$
Call TM of H10B on each of
$p(x,y,z), \quad p(x,y,-z), \quad p(x,-y,z), \quad p(x,-y,-z)$
$p(-x,y,z), \quad p(-x,y,-z), \quad p(-x,-y,z), \quad p(-x,-y,-z)$
And accept if TM for H10B accepts on any, otherwise answer no.

H10A $\leq_m$ H10B
p has an integer solution
$\Leftrightarrow$
q has a nonnegative integer solution
$q = p(x,y,z)p(x,y,-z)\dots p(-x,-y,-z)$

INT2CFG Reduction
G_1: $S_1 \to t_i S_1 i | t_i \# i \quad \forall i, \quad 1 \leq i \leq k$
G_2: $S_2 \to b_i S_2 i | b_i \# i \quad \forall i, \quad 1 \leq i \leq k$
So PCP $\leq_m$ INT2CFG

AMBIG Reduction
$I \xrightarrow{f} G$, I has a match iff G is ambiguous
$G: S \to S_1 | S_2$ where S_1, S_2 as above

Problems in Logic

March-12-13 11:09 AM

Church

The theory of natural numbers with $+$ and $\times$, $Th(\mathbb{N}, +, \times)$ is not recursively solvable.
There is no algorithm, that, given a sentence in this logical theory, will either produce a proof or say no such proof exists.

Theory $Th(\mathbb{N}, +, <)$

Allowed:

$x + y = z?$
$x < y?$
$x = y?$
$\forall, \exists, \ \wedge, \vee, \Rightarrow, \neg$

Chicken McNuggets Theorem

Every integer $N > 43$ can be obtained at McDonald's as the number of McNuggets if one buys pack of 6, 8, and 20 only. Furthermore, 43 is the smallest such.

In $Th(\mathbb{N}, +, <)$

Let $na = a + a + a + \cdots + a + a$ (total of n times)

$(\forall N > 43, \exists a, b, c\ s.t. N$
$= a + a + a + a + a + a + b + b + \cdots (9 \text{ times}) \cdots + b + c$
$+ \cdots (20 \text{ times}) + c) \wedge \left(\forall a', b', c', \neg(43 = 6a' + 9b' + 20c')\right)$

Prefix normal form

[quantifiers] ["atomic" formula involving variables & $+$ & logical operators & $<$ ...]

Typical sentence (assume variables are over $\mathbb{N}$)

$\forall n, a, b, c\ \ (n > 2) \wedge (a, b, c > 0) \Rightarrow a^n + b^n \neq c^n$

Example: Chicken McNuggets Theorem in Prefix Normal Form

$\forall N\ \exists a\ \exists b\ \exists c\ \forall a' \forall b'\ \forall c'\ \left((N > 43) \Rightarrow (N = 6a + 9b + 20c)\right)$
$\wedge \neg(43 = 6a' + 9b' + 20c')$

A Decision Procedure for $Th(\mathbb{N}, +, <)$

Main ideas

- represent possible variables as strings in base 2
- model a formula by a series of automata where the automata accept strings representing possible values of variables that make the formula true.

So our formula looks like
$F = Q_1 x_1\ Q_2 x_2\ \ldots Q_n x_n\ \ \psi$
ψ: atomic formula in $x_1, \ldots, x_n$
$\phi_i = Q_{i+1} x_{i+1} \ldots Q_n x_n \psi$
$\phi_n = \psi, \qquad \psi_0 = F$

Input symbols are n-tuples corresponding to a character in the strings of $(x_1, x_2, \ldots, x_n)$.

Want to build automaton A_n to accept input for which ϕ_n is true.
Given A_n build A_{n-1}
Case 1: $\psi_{n-1} = \exists x_n \psi_n$
Build an NFA (A_{n-1}) that on input representing $(x_1, x_2, \ldots, x_{n-1})$ guesses x_n nondeterministically and checks using A_n if the formula is true.
Case 2: $\psi_{n-1} = \forall_n \psi_n = \neg \exists x_n \neg \psi_n$
Must convert NFA to DFA with subset construction then negate it.

For first quantifier symbol can use usual method of checking if a DFA accepts any/all strings (graph search).

Each time there is an alternation in quantifiers between $\forall$ and $\exists$ must do a subset constructions. With t alternating quantifiers, $\left\lceil \frac{t}{2} \right\rceil$ subset constructions.

$O\left(2^{2^{2^{\cdots 2^u}}}\right)$

Stack of 2's is $\left\lceil \frac{t}{2} \right\rceil$ high. u is length of ψ

Time Complexity

March-14-13 10:36 AM

Our computing model for time bounds will be the multitape TM.

$\text{TIME}\big(f(n)\big) =$ class of languages accepted by
multitape TMs in time $O\big(f(n)\big)$

$\{0^n 1^n : n \geq 0\} \in \text{TIME}(n)$

Theorem

If L is accepted by a multitape TM in $O\big(f(n)\big)$ time, and $f(n) \in \Omega(\sqrt{n})$ then it is accepted by a 1-tape TM in $O(f(n)^2)$ time.

Polynomial Time Decidable Languages

$P = \bigcup_{k \geq 1} TIME\big(n^k\big)$

$= \big\{L : \exists a\ TM$ deciding L & k with worst-case running time $O\big(n^k\big)\big\}$

What can we compute with bounds on resources?

- time
- space
- randomness

Time

Put time bounds on our TM's

Example

$\{0^n 1^n : n \geq 0\}$

The model used is important

input is of length N

how many steps? (transitions of a TM)

$t(N) =$ worst-case # of steps over all inputs of length N

1-Tape TM

Go back and fourth crossing off symbols $t(N) = \Theta(N^2)$

Can we do better?

1. Check that the input is of the form $0^n 1^m$ for some n, m: $O(N)$
2. Check parity of tape. Reject if parity 1.
3. Cross off every other 0
4. Cross off every other 1
5. Check if tape has no 0s and no 1s left. If so accept
    a. Goto 2

$t(N) = \Theta(N \log N)$

Best possible for this problem for 1-Tape TM

2-Tape TM

Copy 1's onto 2nd tape and compare.

$t(N) = \Theta(N)$, best possible

Moral

The choice of computing model affects the running time achievable for a model.

But not too much if the model is reasonable.

Proof

Recall how to simulate a multitape TM with a 1-tape TM

Have many tracks, where pairs simulate a tape by having a tape and pointer.

Each step of the TM costs $O\big(f(n)\big)$ (may have to walk distance $f(n)$ down each track)

$O\big(f(n)\big)$ steps for a total of $O(f(n)^2)$

But have to initialize tape first which takes $O(n)$ time. So $t(n) = O(f(n)^2 + n)$

P and NP

March-19-13 10:04 AM

P

P = class of languages where we can decide membership in polynomial time in $|w|$ where w is input.

NP

NP = a class of decision problems decided by nondeterministic TM' s running in polynomial time.

The runtime of a nondeterministic TM:

The longest computational path is of length $\leq Cn^k$

Equivalently,
NP = a class of decision problems where membership is efficiently checkable given some extra information, called a **certificate**.

Verifier

Polynomial-Time Verifier for L:
An algorithm A, running in polynomial time, that takes inputs of the form $\langle w, c\rangle$ where you are checking if $w \in L$. c is a string ("certificate")
$L = \{w : \exists c \ s.t. \ A \text{ accepts } \langle w, c\rangle\}$
think of c as a way to convince someone that $w \in L$
$|c|$ must be polynomial in $|w|$

P = NP?

$1,000,000 question (Clay Mathematical Institute)

Co-NP

Co-NP $= \{L : \bar{L} \in NP\}$

Polynomial-Time Reductions

$L_1 \leq_P L_2$ means $\exists$ a polynomial-time computable function f such that
$x \in L_1 \Leftrightarrow f(x) \in L_2$

$$x \rightarrow \boxed{\rightarrow \boxed{f} \xrightarrow{f(x)=y} \boxed{Is\ y \in L_2} < \begin{matrix} \text{yes} \\ \text{no} \end{matrix}} \begin{matrix} \rightarrow \\ \rightarrow \end{matrix}$$

NP-Complete (NPC)

The class of languages L in NP such that
$\forall L' \in NP, \quad L' \leq_p L$
Cook proved SAT $\in$ NPC
Karp proved: Many other problems (e.g. HAM-CYCLE) are in NPC

Theorem

If $L_1 \leq_p L_2$ and L_1 is NP-Complete and L_2 is in NP then L_2 is NP-Complete

NP-Hard

We say a language is NP-hard if $\forall L' \in$ NP, $L' \leq_P L$

L NP-complete $\Leftrightarrow$ $\begin{matrix} L \text{ is NP-hard} \\ L \text{ is NP} \end{matrix}$

P = class of languages where we can decide membership in polynomial time in $|w|$ where w is input.
More informally,

- the class of problems with polynomial-time solutions.

Objections:

1. $\Theta(n^{1000})$ time is not realistic
2. $\Theta(n^{\log\log\log n})$ is not in P
   But for all practical purposes this is $O(n^3)$
3. Constant in front ignored

Not all important problems seem to be in P
Some are provably not - ERE universality problem
ERE = extended regular expression = regexp + exponentiation to an integer
Universality problem: $L(\epsilon) = \Sigma^*$

Hamiltonian Cycle Problem (HAM-CYC)

- Given a graph (undirected, although there exist directed version)
- $\exists$ a cycle in which every vertex is visited at most once.
- This is in NP

Equivalence of NP Definitions

If have nondeterministic TM, let the certificate be the choice of decisions during the computation. Can verify on deterministic TM in polynomial time by taking those decision choices.

If have verifier, $|c|$ is bounded by a polynomial in $|p|$ so nondeterministically generate all possible c up to that bound and test in polynomial time.

Example of Polynomial-Time Reductions

PRIMALITY $\leq_P$ PRIME-FACTORIZATION
ELEMENT DISTINCTNESS $\leq_P$ SORTING

Not a polynomial-time reduction:
H10A $\leq_P$? H10B
mapping
$f(x,y,z) \rightarrow f(x,y,z)f(x,y,-z)f(x,-y,z) \dots f(-x,-y,-z)$
generates an exponential-sized output.

SAT

Boolean Satisfiability
Given a Boolean expression consisting of

- literals - variables or negations
- Boolean operators $(\wedge, \vee, \neg, \Rightarrow, \dots)$

is there some assignment of truth values to variables that make the expression true?
e.g. $(x \vee \bar{y} \vee \bar{z}) \Rightarrow \neg(x \wedge y) \vee (y \wedge z)$

CNFSAT

Boolean formula in conjunctive normal form (CNF)
$C_1 \wedge C_2 \wedge \cdots \wedge C_n$
$C_i = (l_1 \vee l_2 \vee \cdots \vee l_{k_i})$
"AND of OR's"

3SAT

CNFSAT with precisely 3 literals per clause (usually distinct)

Proof of Theorem

Let $L \in NP$
Then $L \leq_P L_1$, but $L_1 \leq_P L_2$
$x \in L \Leftrightarrow f(x) \in L_1$
$y \in L_1 \Leftrightarrow g(y) \in L_2$
Take $y = f(x)$ then
$x \in L \Leftrightarrow g(f(x)) \in L_2$

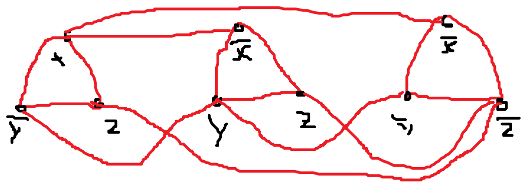
If g is $O(n^k)$ and f is $O(n^l)$ then $g \circ f = O(n^{kl})$ so
$L_1 \leq_P L_2$

Independent Set Problem

INDEP SET $= \{(G, k) : G$ has an independent set of size $k\}$
An independent set of G is a subset of the vertices, no 2 of which are connected by an edge.
3SAT $\leq_P$ INDEP SET

Make each clause into a triangle in the graph. Connect vertices that represent negated variables.
$(x \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$

$(x \vee y \vee z) \wedge (x \vee y \vee z) \wedge (x \vee y \vee z)$



Claim: The graph has an independent set of size k iff the Boolean expression is satisfiable.

If expression is satisfiable, select one node corresponding to on true variable/negated variable in each clause.
Conversely, an independent set of size k (where k is the number of clauses) provides a valid assignment for the expression.

Note also that INDEP SET is in NP. It is easy to verify.

SAT is NP-Complete

March-21-13 10:05 AM

Theorem (Cook, Levin)

The problem SAT is NP-Complete.

Theorem

CNFSAT is NP-Complete

Theorem

3SAT is NP-Complete

Clique Problem

Instance

Undirected graph $G = (V, E)$ and an integer k

Question

Does G have a subset $V' \subseteq V$ of cardinality k such that every two distinct vertices in V' are connected by an edge.

SUBSET SUM

Instance

Given a set of non-negative integers $S = \{x_1, x_2, \dots, x_t\}$ and a target T

Question

Does there exist a subset of S whose sum is T

Proof of Theorem (SAT is NP-Complete)

SAT ∈ NP

Guess an assignment of truth values for variables and check that the given formula evaluates to true.

$\forall L \in \mathbf{NP}\ L \le_P \mathbf{SAT}$

Have TM M for L

On input x want to know if $x \in L$

$x \xrightarrow{\text{poly } f} \phi_x$

$x \in L \Rightarrow \phi_x$ is satisfiable

ϕ must mimic the computations of M on input x

- ensure M starts in right configuration
- ensure that M follows its own rules
- ensure that M reaches q_{accept} iff $x \in L$
- must not be too big (in $|x|$)
    - Can be any size in terms of M

Write as a square array storing the state of the TM in each row

Step	Col. 1	Col. 2	Col. 3	...						
1	#	q_0	x_1	x_2	x_3	...	x_n	␣	␣	#
2										#
⋮										⋮
m				q_{accept}						#

M runs in $O(n^k)$ time, $n = |x|$

So the array needs to be at most $Cn^k \times Cn^k$

$\phi_x = \phi_{\text{init}} \wedge \phi_{\text{final}} \wedge \phi_{\text{move}} \wedge \phi_{\text{cell}}$

Variables:

$x_{i,j,s} = \text{true} \Leftrightarrow$ cell in row i and column j has symbol a_s in it where the cells contain $\{a_1, a_2, \dots, a_t\} = \Gamma \cup Q \cup \{\#\}$

First row is correct:

$\phi_{\text{init}} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,x_1} \wedge \cdots \wedge x_{1,n+2,x_n} \wedge x_{1,n+3,␣} \wedge \cdots \wedge x_{1,Cn^k-1,␣} \wedge x_{1,Cn^k,\#}$

Final row contains an accepting state (if final state is reached prematurely, allow same row to carry forward):

$\phi_{\text{final}} = x_{Cn^k,1,q_{\text{acc}}} \vee x_{Cn^k,2,q_{\text{acc}}} \vee \cdots \vee x_{Cn^k,Cn^k,q_{\text{acc}}}$

Ensure cell validity (each contains exactly one symbol)

$$\phi_{\text{cell}} = \bigwedge_{1 \le i,j \le Cn^k} \left(\bigvee_{1 \le r \le t} x_{i,j,a_r} \wedge \bigwedge_{\substack{r \ne s \\ 1 \le r,s \le t}} \neg\left(x_{i,j,a_r} \wedge x_{i,j,a_s}\right) \right)$$

Ensure valid moves

Need to look at groups of 3 cells in row i and compare with row $i+1$

$$\phi_{\text{move}} = \bigwedge_{\substack{1 \le i \le Cn^k-1 \\ 1 \le j \le Cn^k-2}} LEGAL(i,j)$$

$LEGAL(i,j) =$ windows whose upper left is at (i,j) is legal.

window looks like

b_1	b_2	b_3
b_4	b_5	b_6

$$LEGAL(i,j) = \bigvee_{(b_1,b_2,b_3,b_4,b_5,b_6) \in LE} \left(x_{i,j,b_1} \wedge x_{i,j+1,b_2} \wedge x_{i,j+2,b_3} \wedge x_{i+1,j,b_4} \wedge x_{i+1,j+1,b_5} \wedge x_{i+1,j+2,b_6}\right)$$

LE is the set of all legal 6-tuples. This is a finite set that depends on M

Check Sizes

$|\phi_{\text{init}}| \in O(n^k)$

$|\phi_{\text{final}}| \in O(n^k)$

$|\phi_{\text{cell}}| \in O(n^{2k})$

$|\phi_{\text{move}}| \in O(n^{2k})$

So this formula is polynomial in $|x|$

■

Theorem (CNFSAT is NP-Complete)

Modify construction from above proof.

ϕ_{init} is an $\wedge$ of clauses

ϕ_{final} is a single clause

ϕ_{cell} can be rewritten

$$\phi_{\text{cell}} = \bigwedge_{1\le i,j\le Cn^k} \left(\bigvee_{1\le r\le t} x_{1,j,a_r} \wedge \bigwedge_{\substack{r\neq s \\ 1\le r,s\le t}} \left(\overline{x_{i,j,a_r}} \vee \overline{x_{i,j,a_s}}\right) \right)$$

so ϕ_{cell} is now an $\wedge$ of clauses

ϕ_{move} is $\wedge$ of $\vee$ of $\wedge$
Use distributive property: $(x_1 \wedge x_2) \vee (x_3 \wedge x_4) = (x_1 \vee x_3) \wedge (x_1 \vee x_4) \wedge (x_2 \vee x_3) \wedge (x_2 \vee x_4)$
So $LEGAL(i,j)$ can be written as $\wedge$ of $\vee$s. This may make it very big but size is in terms of M so not a problem.

Therefore, by the same argument, CNFSAT is NP-Complete. ■

Theorem (3SAT is NP-Complete)

CNF Formula: $C_1 \wedge C_2 \wedge \cdots \wedge C_r$
each C_i looks like $l_1 \vee l_2 \vee \cdots \vee l_j$
Two bad cases:
$j < 3$:
Introduce new global variables t and u

$l \to (l \vee t \vee u) \wedge (l \vee \bar{t} \vee u) \wedge (l \vee t \vee \bar{u}) \wedge (l \vee \bar{t} \vee \bar{u})$
$(l \vee m) \to (l \vee m \vee t) \wedge (l \vee m \vee \bar{t})$
$j > 3$:
Introduce new variables to chain:
e.g.
$(l_1 \vee l_2 \vee l_3 \vee l_3 \vee l_4 \vee l_5) \to (l_1 \vee l_2 \vee m_1) \to (\overline{m_1} \vee l_3 \vee m_2) \wedge (\overline{m_2} \vee l_4 \vee l_5)$
Each of these increases the size at most linearly so good.
Therefore CNFSAT $\le_P$ 3SAT
■

Clique Problem is NP-Complete

INDEP SET $\le$ CLIQUE
$(G,k) \to (G',k')$
$G' = (V, \bar{E})$, Complementary graph
$k' = k$

If G has n nodes, $|\bar{E}| \le n^2$
Note to self:
What if only specified graph using edges with nodes implicitly numbered 1-n?
Answer: In most graph algorithms, could just ignore vertices with no edges. In this case, don't include those in G' and then just add t to the maximum clique size, where t is the number of vertices with no edges.
■

SUBSET SUM Problem is NP-Complete

Will show 3SAT $\le_P$ SUBSET SUM

$\phi \to S, T$
variables $x_1, x_2, \dots, x_l$ $\quad \Rightarrow S = \{y_1 z_1, y_2, z_2, \dots, y_l, z_l,$
clauses $c_1, c_2, \dots, c_k$ $\quad g_1, h_1, \dots, g_k, h_k\}$

Each value in S is a decimal number containing only 0 and 1
digit c_j in $y_i = 1 \Leftrightarrow c_j$ contains x_i
digit c_j in $z_i = 1 \Leftrightarrow c_j$ contains $\bar{x}_i$
g_i and h_i are slack variables. Digit $c_j = 1$ in $g_i, h_i \Leftrightarrow i = j$

	1	2	3	...	l	c_1	...	c_k
y_1	1	0	0	...	0		...	
z_1	1	0	0	...	0		...	
y_2	0	1	0	...	0		...	
z_2	0	1	0	...	0		...	
y_3	0	0	1	...	0			
z_3	0	0	1	...	0			
$\vdots$								
g_1						1		
h_1						1		
g_2								
h_2								
$\vdots$								
T	1	1	1	...	1	3	3	3

There is a subset of these summing up to T iff there is a valid assignment of the variables in ϕ.
See Siper for a description of this reduction.

Space Complexity

March-26-13 10:03 AM

The space complexity of a deterministic TM that halts on all inputs is
$f(n) = \max_{|w|=n}(\#\text{ tape cells used on input } w)$

For a nondeterministic TM that halts on all computational paths on all inputs:
$f(n) = \max_{|w|=n}(\max\ \#\text{ of tape cells used on any comp. path for } w)$

SPACE

also called DSPACE - deterministic
$\text{SPACE}(f(n)) = \text{DSPACE}(f(n))$
$= \{L : L \text{ is decided by an } O(f(n))\text{-space-bounded deterministic } TM\}$

NSPACE - nondeterministic
$\text{NSPACE}(f(n))$
$= \{L : L \text{ is decided by an } O(f(n))\text{-space-bounded nondeterministic } TM\}$

PSPACE

$$\text{PSPACE} = \bigcup_{k\geq 1} \text{SPACE}(n^k)$$

EXPTIME

$$\text{EXPTIME} = \bigcup_{k\geq 1} \text{DTIME}\left(2^{O(n^k)}\right)$$

We know P ≠ EXPTIME

$\boxed{\text{EXPTIME}\ \boxed{\text{PSPACE}\ \boxed{\text{NP}\ \boxed{\text{P}}}}}$

Don't know whether EXPTIME ≠ PSPACE, PSPACE ≠ NP, and/or NP ≠ P

Theorem

If $L \in \text{NTIME}(f(n))$ and $f(n) \geq n$ then
$L \in \text{DTIME}(2^{cf(n)})$

Savitch's Theorem

If $L \in \text{NSPACE}(f(n))$ and $f(n) \geq n$ then $L \in \text{DSPACE}(f(n^2))$

Implication

NPSPACE = PSPACE

LENGTH-UNIVERSALITY PROBLEM FOR NFA's

Instance:

An NFA of n states.

Question:

Does there exist some length L such that M accepts all strings of length L.

Is LENGTH-UNIVERSALITY FOR NFA's in PSPACE?
Unsolved.

Example

$\text{SAT} \in \text{SPACE}(n)$

Proof

Try each possible assignment of variables.
Use binary counter, $O(n)$ space
Can evaluate each expression in $O(n)$ space

Example

NUP: NON-UNIVERSAILITY PROBLEM FOR NFA's

Instance

An NFA A over an alphabet Σ

Question

Is $L(A) \neq \Sigma^*$

NUP $\in \text{NSPACE}(n)$
Algorithm: Nondeterministic "guess" $w \neq L(A)$ and check it.

If A has n states then $L(A) \neq \Sigma^*$ iff A rejects a string of length $\leq$?
Take NFA A of n states, convert to DFA A' of $\leq 2^n$ states.
If A' doesn't accept w is doesn't accept w' of length at most $2^n - 1$

$L(A) \neq \Sigma^* \Leftrightarrow \exists w \notin L(A), \quad |w| < 2^n$
Need $O(n)$ for counter
$O(n)$ to maintain list of states for the NFA

Proof of Theorem

Construct computational tree for input w of length n and traverse it in breadth-first search.
$f(n)$ nondeterministic choices, $r = 2^c$ is max branching factor.
The tree is of size at most $O(r^{f(n)})$. Takes $O(r^{f(n)}) = O(2^{cf(n)})$ time to traverse.

Proof of Savitch's Theorem

Suppose L is accepted by NTM N running in $f(n)$ space.
$N's$ running time $\leq 2^{cf(n)}$
Configuration: xqy

tape has xy, state q, scanning 1st symbol of y
xy can be $\leq f(n)$

Idea: a big graph G of possible moves of N on input w
Each vertex is a configuration. Edge from one to another if possible to go from that configuration to the next.

Want to find path between two vertices.
$\text{CANYIELD}(c_1, c_2, t) =$ true iff there is a choice of
$\leq t$ moves that takes me from configuration c_1 to configuration c_2.

Idea:
$c_1 \xrightarrow{\leq t} c_2 \Leftrightarrow c_1 \xrightarrow{\leq \frac{t}{2}} c_m \xrightarrow{\leq \frac{t}{2}} c_2$

```
CANYEILD(c₁, c₂, t)
    if t ≤ 1:
        check if c₁ = c₂ (0 moves)
        or c₁ ⊢ᴺ c₂ (1 move of N)
            accept if true
    otherwise:
        for each possible cₘ (≤ f(n)C^f(n) = O(2^df(n)))
            CANYEILD(c₁, cₘ, ⌊t/2⌋)
            CANYEILD(cₘ, c₂, ⌈t/2⌉)
            if both return true, return true.
    otherwise reject
```

Each CANYIELD call has a stack frame size of $O(f(n))$ bits.
Recursion depth is $O(\log t) = O(\log 2^{cf(n)}) = O(f(n))$

Total space is $O\left((f(n))^2\right)$
Call $\text{CANYEILD}(c_{\text{init}}, c_{\text{accept}}, 2^{cf(n)})$
Assume that if N accepts, it erases its tape, moves head to the left, and enters q_{accept}. So there is only one accepting configuration to check.

But don't necessarily know $f(n)$.
Call $\text{CANYEILD}(c_{\text{init}}, c_{\text{accept}}, i)$ for $i = 1, 2, 3, \ldots$

when at step i, go through all configurations c of size $i + 1$
check $\text{CANYIELD}(c_{\text{init}}, c, i + 1)$.

If all fail, halt.

This is a minor point because usually know $f(n)$

PSPACE-Complete

March-28-13 10:02 AM

True Quantifiable Boolean Formula (TQBF)

- Boolean formulas, like in SAT ≡ logical connectives and literals
- add ∃ and ∀
- quantifiers come first (prenex normal form)
  - $Q_1 x_1 Q_2 x_2 \ldots Q_i x_i \phi$

TQBF is a generalization of SAT. SAT is
$\exists x_1 \exists x_2 \cdots \exists x_i \phi$

PSPACE-Complete

L is PSPACE-Complete if
1) $L \in$ PSPACE
2) $\forall L' \in$ PSPACE, $L' \leq_P L$
   Reduce in polynomial time, not space.

If just 2) is true, L is PSPACE-hard

Theorem

TQBF is PSPACE-complete

FORMULA GAME

Assume ∃ and ∀ alternate (can stick in dummy variables to make it so)
Think of it as a game:
∃ - Edward - trying to make formula true
∀ - Alice - trying to make formula false
FORMULA GAME:
Given a formula ϕ, does Edward have a winning strategy? Makes the formula true.

GENERALIZED GEOGRAPHY

Given a directed graph G and an initial vertex v, players Edward and Alice take turns choosing a previously unvisited vertex connected to the current one.

Question:

Does Edward have a forced win?

Proof of Theorem

Draw tableau of tape configurations for PSPACE computation
Width: polynomial
Height: c^{n^k}, $c = |\Gamma| + |Q|$
So this strategy fails. Can't prove by same method as SAT is NP-Complete

Idea:
Create a Boolean formula $\phi_{c_{\text{init}}, c_{\text{accept}}, t}$, $t \approx 2^{c' n^k}$ for some c', k

Show how to build $\phi_{c_1, c_2, t}$
What are the variables? Each cell of configuration has a variable
abqcde
x_{i, a_j} = true iff symbol i of the configuration = a_j
Write formula to verify each cell has exactly one variable.

Base case:
$t = 0$: True if both rows of the same
$t = 1$: Allow a single transition. Like SAT proof.

Try divide and conquer:
$\phi_{c_1, c_2, t} = \exists c_m \, \phi_{c_1, c_m, \frac{t}{2}} \wedge \phi_{c_m, c_2, \frac{t}{2}}$
Recurrence is $|\phi_t| = 2\left|\phi_{\frac{t}{2}}\right|$
Gives $|\phi_t| \approx t \log t$
t is exponentially large, so this does not work.

Want to avoid multiplying formula. Write something like
$\phi_{c_1, c_2, t} = \exists c_m \, \forall (c, d) \in \{(c_1, c_m), (c_m, c_2)\} \; \phi_{c, d, \frac{t}{2}}$
More formally,
$\phi_{c_1, c_2, t} = \exists c_m \forall c \, \forall d \, \big((c = c_1 \wedge d = c_m) \vee (c = c_m \wedge d = c_2)\big) \Rightarrow \phi_{c, d, \frac{t}{2}}$
Note, this requires t to be a power of two. Just allow an accepting configuration to repeat so there will be a path in a power of 2 number of steps.
Also assume only one accepting configuration.

$|\phi_{c_1, c_2, t}| \approx \log t \approx c' n^k$, is polynomial.
Therefore, TQBF is PSPACE-hard

All variables are boolean, so recursive assign 0 or 1 to each variable and check if ∀ or ∃ is true. Stack is $O(n)$ in size. Therefore TQBF is in PSPACE
∴ TQBF is PSPACE-Complete

GENERALIZED GEOGRAPHY in PSPACE-Complete

In PSPACE. Can solve it in PSPACE the same way as TQBF

See Sipser for reduction from TQBF

LOGSPACE

April-02-13 10:02 AM

LOGSPACE

L = DLOGSPACE = DSPACE(log n)
NL = NLOGSPACE = NSPACE(log n)

A pointer into the input requires log n space so can think of this as the problems that can be solved with a constant number of pointers.

Theorem

$L \subseteq P$
$NL \subseteq P$

Theorem

PATH ∈ NL

Corollary, by an appropriately modified Savitch's theorem for this model,
PATH ∈ DSPACE$\left((\log n)^2\right)$

Open Problem

Does L = NL?

NL-Complete

A is NL complete if

1) A ∈ NL
2) For all B ∈ NL, $B \leq_L A$

$\leq_L$ is a logspace reduction (uses a logspace transducer)

Hierarchy:
$\boxed{\boxed{L}\ NL\ \boxed{NLC}}$
Non-intersection of L and NLC is hypothesized but unknown.

Logspace Transducer

A logspace-transducer is a TM T with 3 tapes:

- a read-only input tape
- a write-only output tape where the head can only move right
- a work tape using only $O(\log n)$ cells

On input x, the machine T write $f(x)$ on its output tape.
$\xrightarrow{x}\boxed{T}\xrightarrow{f(x)}$

Theorem

PATH is NL-Complete

Theorem

If $A \leq_L B$ and $B \in L$ then $A \in L$
If $A \leq_L B$ and $B \in NL$ then $A \in NL$

Theorem

NL ⊆ P

Hierarchy of Complexity

$$\boxed{\begin{array}{c} PSPACE \\ \boxed{\begin{array}{c} NP \\ \boxed{\begin{array}{c} P \\ \boxed{\begin{array}{c} NL \\ \boxed{L} \end{array}} \end{array}} \end{array}} \end{array}}$$

Sublinear Space Model

Need a new model to deal with sublinear space. With TM we get linear space for free.
Input tape is finite and contains just the input with delimiters.

#	i	n	p	u	t	$

The input tape is read only.
There is a read/write work tape, and a finite control.

The space used on an input of size n = maximum number of cells scanned on the work tape.

Proof of Theorem (L∈ P)

A configuration of a machine running in log space:

- Contents of work tape ≤ c log n cells
  - each cell holds an element of Γ, the tape alphabet: $|\Gamma|^{c\log n}$
  - state currently in: $|Q|$ possibilities
- head position on input tape: n+2 possibilities
- head position on work tape: c log n

Total # of configurations:
$\left(n^{c\log|\Gamma|}\right)|Q|(n+2)(c\log n) \leq Cn^e, \qquad e = (c\log|\Gamma|)+2$
simulate M, keeping track of # configurations.
If it exceeds Cn^e, then in an infinite loop so halt and reject, otherwise do what M does.

The same argument applies for $NL \subseteq NP$

Example

$A = \{0^n 1^n : n \geq 0\}$
$A \in L$?
Yes, because we can use the work tape as a binary counter. $O(\log n)$ bits to count # of 0's
Use decrementer for each 1. Hit 0, accept if at right end marker

Example: PATH

PATH = $\{\langle G, s, t\rangle : G$ is a directed graph with a directed path from s to $t\}$
PATH ∈ P? Do a search (BFS/DFS)
PATH ∈ L? Nobody knows! Undirected PATH (USTCON ∈ L by Reingold 2008)

Theorem (PATH ∈ NL)

G is a graph with $M = |V|$ vertices
Start at s,

    u := s
    counter := 0
    while counter < M
        if v = t then accept
        guess nondeterministically an edge (u, v)
        if it exists, replace u by v
        counter++
    else reject

Total space needed is $O(\log n)$, n = input size

Proof of Theorem (PATH is NL-Complete)

1) Done
2) Take a language $B \in NL$
   Then there exists a nondeterministic $O(\log n)$-space bounded TM M accepting B

   Given the description of M and input w, we need to output G, the configuration graph and vertices s, t such that ∃ a directed path $s \to t$ in G iff M accepts w.
   A configuration of M - head positions, work tape contents, state
   We can check if $c_1 \vdash c_2$ in log space.
   s = initial configuration, t = accepting configuration.

Recall:
$A \leq_P B$ and $B \in P$ then $A \in P$

If $f(x)$ is computable in $\leq c|x|^k$ time and if B-decider runs in , $= c'|y|^l$ time then
$x \boxed{\rightarrow \boxed{f} \xrightarrow{f(x)=y} \boxed{B-decider} <} \begin{array}{l} \text{yes} \\ \text{no} \end{array}$
runs in $c|x|^k$ time to compute $f(x)$, $|f(x)| \leq c|x|^k + c'|y|^l$, $y = f(x) \leq c'\left(c|x|^k\right)^l$
Total time: $c|x|^k + c'c^l|x|^{kl} = O\left(|x|^{kl}\right)$

Proof of Theorem

Machine for A simulates the machine computing f but discards all output bits except the one that is needed by B. That one is given by a counter into the input tape which "holds" $f(x) = y$

Each time simulator for B needs an input cell, run f on A to get it.

Proof of Theorem (NL $\in P$)

Let A ∈ NL, then $A \leq_L$ PATH. So $A \leq_P$ PATH

We know PATH $\in$ P using BFS or DFS
Therefore, A $\in$ P

Co-NL

April-04-13 10:05 AM

Theorem

NL = co-NL
co-NL $= \{A : \bar{A} \in NL\}$

Observation

$A \leq_L B \Rightarrow \bar{A} \leq_L \bar{B}$

Proof:

Use the same logspace transducer

Lemma

$A \leq_L B$ and $B \in$ co-NL
then A ∈ co-NL

Proof of Theorem

Strategy: $\overline{\text{PATH}} \in$ NL
PATH $= \{\langle G, s, t\rangle : G$ is a directed graph and there is a path from s to $t\}$

Goal: given a graph G and vertices *s,t* determine that there is no path from s to t in G in nondeterministic log space.

Note that $\overline{\text{PATH}}$ consists of malformed inputs, and inputs representing $\langle G, s, t\rangle$ where s is not reachable from t. Former case can be done in not much space.

Proof of Lemma

B ∈ co-NL $\Rightarrow \bar{B} \in$ NL
if $A \leq_L B$ then $\bar{A} \leq_L \bar{B}$ so $\bar{A} \in$ NL
Then A ∈ co-NL ■

Why is proving $\overline{\text{PATH}} \in$ NL sufficient?

Now PATH is NL-complete
Let A be a language in NL
A $\leq_L$ PATH $\Rightarrow \bar{A} \leq_L \overline{\text{PATH}}$
if $\overline{\text{PATH}} \in$ NL then $\bar{A} \in$ NL so $A \in$ co-NL

How to verify in NL that no $s \to t$ path exists, given C that is the total number of nodes reachable from S.

1. Loop over all vertices v
    a. Guess a path $s \to t$ and verify it.
        i. If path exists and $v = t$ then reject
        ii. If path exists and $v \neq t$ then $ctr \coloneqq ctr + 1$
        iii. If path does not exist reject
    b. if $ctr = C$ then accept

Compute for each i, $0 \leq i < |V|$
how many vertices are reachable from s in G in i steps.
$A_i = \{v \in V : v$ reachable from s in i steps$\}$

$C_0 = 1, \quad A_0 = \{s\}, \quad A_i \subseteq A_{i+1}$
How to compute C_{i+1} from C_i?
S := 1
For each vertex u (want to find $i+1$ step path from v to u)
T := 0 (enumerates vertices in A_i)
For each vertex v, guess an i-step path from s to v
if successful, increment counter T
check if $u = v$ or (v, u) is an edge.
If so, increment counter S and break
If $T \neq C_i$, reject (guessed wrong and missed a path in A_i)
$C_{i+1} \coloneqq S$

Hierarchies

April-04-13 10:46 AM

Theorem

If $f(n), g(n)$ are functions from $\mathbb{N} \to \mathbb{N}$ and $f(n) = o(g(n))$ then $\text{DSPACE}(f(n)) \subsetneq \text{DSPACE}(g(n))$

Space-Constructible

$g(n)$ is space-constructible if $\exists$ a TM that on input 1 writes $g(n)$ its output tape and uses $O(g(n))$ space

Proof of Theorem

Idea

Create a TM A accepting $L(A) \in \text{DSPACE}(g(n))$ but no machine running in $f(n)$ space will be able to decide $L(A)$. $g(n)$ must be space-constructible.

A:

- rejects if the input is not $\langle M \rangle 10^i$, for $i \geq 0$
    - By allowing arbitrarily large input, eventually $Cf(n) < g(n)$ for large enough n
- otherwise, it simulates M on input $\langle M \rangle 10^i$ and do the opposite.
    - If M accepts we reject
    - if M rejects we accept
- need the ability to mark $g(n)$ tape cells. Fine so long as $g(n)$ is space constructible
- mark tape at position $g(n)$ and if simulation of M exceeds, reject.
- Use another track to count the number of steps in the simulation of M on $\langle M \rangle 10^i$
    - if it exceeds $2^{g(n)}$, reject.

So A behaves differently than every TM for $\text{DSPACE}(f(n))$
Therefore $\text{DSPACE}(f(n)) \neq \text{DSPACE}(g(n))$

Definitions

Regular Expression: $\Delta = \{\cup, *, (,)\} \cup \Sigma \cup \{\epsilon, \emptyset\}$
DFA: $M = (Q, \Sigma, \delta, q_0, F)$, $L(M) = \{x \in \Sigma^* : \delta(q_0, x) \in F\}$
NFA: $M = (Q, \Sigma, \delta, q_0, F), \delta : Q \times \Sigma \to \mathcal{P}(Q)$, $L(M) = \{x \in \Sigma^* : \delta(q_0, x) \cap F \neq \emptyset\}$
CFG: $G = (V, \Sigma, P, S)$, $L(G) = \{x \in \Sigma^* : S \Rightarrow^* x\}$
CNF: Every production: $A \to BC$, $A \to a$
PDA: $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$, $\delta : Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \to 2^{Q \times (\Gamma \cup \{\epsilon\})}$
$L(M) = \{x \in \Sigma^* : (q_0, x, \epsilon) \vdash^* (q, \epsilon, \alpha)$ for $q \in F, \alpha \in \Gamma^*\}$
TM: $M = (Q, \Sigma, \Gamma, \delta, q_0, p_r)$, $\delta : (Q - \{q_a, q_r\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$
Accepted/recognized: $L(M) = \{x \in \Sigma^* : q_0 x \vdash^* y q_a z$ for some $y, z \in \Gamma^*\}$
TM M computes a function $f(x)$ if $q_0 x \vdash^* h_a f(x)$, $f : \Sigma^* \to \Gamma^*$

Pumping Lemma for DFA

If L is regular then
$\exists$ a constant $n \geq 1$ (which could depend on L)
$\forall z \in L, \quad |z| \geq n$
$\exists u, v, w$ such that $z = uvw$, $|uv| \leq n$, $|v| \geq 1$
$\forall i \geq 0 \;\; uv^i w \in L$

Contrapositive for PL for DFA

If $\forall$ constants $n \geq 1$
$\exists\, z \in L, \quad |z| \geq n$
$\forall u, v, w$ such that $z = uvw$, $|uv| \leq n$, $|v| \geq 1$
$\exists i$ s.t. $uv^i w \notin L$
then L is not regular

Pumping Lemma for CFL's

If L is a CFL then
$\exists n \geq 1$
$\forall z \in L, \quad |z| \geq n$
$\exists z = uvwxy, \quad |vwx| \leq n, \quad |vx| \geq 1$
$\forall i \geq 0, \quad uv^i wx^i y \in L$

Contrapositive of Pumping Lemma for CFL's

If $\forall n \geq 1$
$\exists z \in L, |z| \geq n$
$\forall z = uvwxy, \quad |vwx| \leq n, \quad |vx| \geq 1$
$\exists i \geq 0$ such that $uv^i wx^i y \notin L$
then L is not a CFL

Closures

$L^2 \subseteq L \Rightarrow L^+ \subseteq L$
$L \in REG \Rightarrow L^R \in REG$
$L \in REG \Rightarrow Pref(L) \in REG$
n-state DFA $\Rightarrow \exists x \in L(M), |x| < n$
& $|L| = \infty \Leftrightarrow \exists x \in L(M), n \leq |x| < 2n$
$L_1, L_2 \in CFL \Rightarrow L_1 \cup L_2, L_1 L_2, L_1^* \in CFL$

Kleene Theorem

DFA = gNFA = NFA-ϵ = REG

State Elimination

1: Convert to 1 initial, 1 final, no trans to initial, no trans out of final.
2: Add transitions $(r_i)(t)^*(s_j)$

Algorithm for CNF

1. Get rid of useless variables
2. Replace all terminals that appear in a RHS with length ≥ 2
   Introduce $A_a \to a$
3. Shorten RHS in large productions
4. Remove ϵ-productions: $A \to \epsilon$
   For $A \Rightarrow^* \epsilon$, replace A by ϵ
5. Remove unit productions
   For $A \Rightarrow^* B$, $\forall B \to \alpha, |\alpha| \neq 1$ add $A \to \alpha$

CFG=>PDA

Store suffix & sentinel on stack. Match prefix w/ input.

PDA=>CFG

$p = q, A_{pp} \to \epsilon, \quad A_{pq} \to A_{pr} A_{rq} \forall r \in Q$
$A_{pq} \to a A_{rs} b$ if $\delta(p, a, \epsilon) \ni (r, U)$ and $\delta(s, b, U) \ni (q, \epsilon)$

Rice's Theorem

If P is a nontrivial property then the decision problem "Given M, does $L(M)$ have the property P?" is unsolvable.

Theorems

$\exists$ universal TM T_U with $\Sigma = \{0, 1\}$ that on input $\langle T, w \rangle$ will simulate T on w and do what T does on input w.
If L and $\bar{L}$ are both Turing-recognizable (r.e.) then L and $\bar{L}$ are both Turing-decidable (recursive)
L accepted by multitape TM in $O(f(n)), f(n) \geq \sqrt{n}$ time $\Rightarrow$ single tape $O(f(n)^2)$
If $L \in \mathrm{NTIME}(f(n))$ and $f(n) \geq n$ then $L \in \mathrm{DTIME}(2^{cf(n)})$
Savitch: If $L \in \mathrm{NSPACE}(f(n))$ and $f(n) \geq n$ then $L \in \mathrm{DSPACE}(f(n^2))$
$A \leq_L B$ and $B \in$ co-NL then $A \in$ co-NL
If $f(n), g(n)$ are functions from $\mathbb{N} \to \mathbb{N}$ and $f(n) = o(g(n))$ then $\mathrm{DSPACE}(f(n)) \subsetneq \mathrm{DSPACE}(g(n))$

$PSPACE$
NP
P
NL
L

Undecidable

Halts, PCP (dominoes), INT2CFG($L(G_1) \cap L(G_2) \neq \emptyset$?), AMBIG (CFG)

NP

HAM-CYC

NP-complete

SAT, CNFSAT, 3SAT, INDEP-SET, CLIQ, SUBSET-SUM

PSPACE

SAT, NUP (NFA A, $L(A) \neq \Sigma^*$)

PSPACE-complete

TQBF, Generalized Geography

NL-Complete

PATH $= \{\langle G, s, t \rangle : G$ is a directed graph with a directed path from s to $t\}$

Reductions

$\leq_T$ Use TM solving P_2 to solve P_1
$\leq_m$"Mapping", $x \in L_1 \Leftrightarrow f(x) \in L_2$
$\leq_P$ Polytime mapping reduction
$\leq_L$ is a logspace reduction, in/work/out

Complexity Classes

$\mathrm{TIME}(f(n))$ = languages accepted by multitape TM's in $O(f(n))$ time
$\mathrm{P} = \bigcup_{k \geq 1} TIME(n^k)$
NP = given certificate, can verify membership in polytime
Co-NP $= \{L : \bar{L} \in NP\}$
NP-hard: $\{L : \forall L' \in NP, \; L' \leq_p L\}$
NP-complete = NP $\cap$ NP-hard
SPACE(f(n)) = DSPACE(f(n))
NSPACE(f(n))
$\mathrm{PSPACE} = \bigcup_{k \geq 1} \mathrm{SPACE}(n^k)$
$\mathrm{EXPTIME} = \bigcup_{k \geq 1} \mathrm{DTIME}\left(2^{O(n^k)}\right)$
L = DLOGSPACE = DSPACE(log n)
NL = NLOGSPACE = NSPACE(log n)
co-NL $= \{A : \bar{A} \in NL\}$ = NL

Space Constructible

g(n) is space-constructible if $\exists$ a TM that on input 1 writes g(n) its output tape and uses O(g(n)) space

All languages | R.E. | Recursive | CFL | REG | FINITE