COS 126 Lecture 15: Finite-State Automata

Specifying "pattern" for grep can be complex

```
^[^aeiou]*a[^aeiou]*e[^aeiou]*i
          [^aeiou]*o[^aeiou]*u[^aeiou]*$
```

What kinds of patterns can be specified?

match all lines containing an even number?

match all lines containing a prime number?

Which aspects are essential?

Regular Expression

0 or 1

(a)

ab

a+b

a*

where a and b are regular expressions

Ex:

(10)*

(0+011+101+110)*

(01*01*01*)*

Formal Languages

Language:

    a set of strings over a finite alphabet

Every regular expression (RE) describes a language

    (the set of all strings that "match")

Regular Language:

    any language representable by an RE

What languages are regular?

Examples (all but one of the following are regular

- all bit strings
  - that begin with 0 and end with 1
  - whose number of 0's is a multiple of 5
  - with more 1's than 0's
  - with no consecutive 1's

Can cast any computation as a language problem

Start by trying to understand simple languages

Finite State Automata

Simple machine with N states
- start in state 0
- read a bit
- move to new state
  - depends on bit, current state
- stop when last bit read
- ACCEPT if in specified state X
- REJECT otherwise

Also say FSA "recognizes" input string or
"decides" whether input string is
in the language it can recognize

Ex:

(10)*

10101010?

013131313

Ex: odd number of 0’s

0001110?

01011110

FSAs and REs are equivalent
[stay tuned]

An application

"Bounce" filter:

remove isolated 0’s and 1’s in a bitstream

input:

0 1 0 0 0 1 1 0 1 1 1 1

output (one-bit delay)

0* 0 0 0 0 0 1 1 1 1 1 1 1

state	in	out
0	0	0
0	1	0
1	0	0
1	1	1
2	0	1
2	1	1
3	0	0
3	1	1

State interpretations

0: at least two consecutive 0’s

1: seq. of 0’s followed by a 1

2: at least two consecutive 1’s

3: seq. of 1’s followed by a 0

C program to Simulate FSAs

```
#include <stdio.h>

main(int argc, char*argv[])
  {
    int zero[100], one[100]; char c;
    FILE *fsa = fopen(*++argv, "r");
    int state, N;
    for (N = 0; !feof(fsa); N++)
      fscanf(fsa, "%d  %d ", &zero[N], &one[N]);
    state = 0;
    while ((c = getchar()) != EOF)
      if (c == '0') state = zero[state];
                    else state = one[state];
    if (state == 0) printf("Accepted ");
    else printf("Rejected ");
  }
```

Ex: FSA to decide if input is divisible by 3
[magic!?]

S	0	1
0	0	1
1	2	0
2	1	2

```
1 1 1 0 1 0 1 0
0 1 0 1 2 2 1 0
```

A language that is not regular

FSAs can't "count"

Theorem: No finite state machine can decide whether or not its input has the same number of 0's and 1's.

Proof:

Suppose an N-state machine does it.
Give it N+1 0's followed by N+1 1's
Some state must be revisited
Machine would accept the same string
    without the intervening 0's
That string doesn't have the same
    number of 0's and 1's
Contradiction.

Ex:

```
.       0000000000000000011111111111111111
.         x   ...      x
.
.       0000001111111111111111111
.         x
```

Need to consider more powerful machines

Pushdown Automata

Add "memory" to the FSA: a pushdown stack
(amount of memory is potentially infinite)

Simple machine with N states (0 to N-1)
- start in state 0
- read a bit, check bit at top of stack
- depending on current state/input bit/stack bit
  - move to new state
  - push or pop a bit
- stop when last bit read
- ACCEPT if stack empty, REJECT otherwise

PDA for deciding whether input has the same
number of 0's and 1's:

PDA's can count, but not much else:
- can't recognize set of all strings of the form AB where both A and B have the same number of 0's and 1's
- more powerful machines needed

Nondeterministic Machines

Add ability to guess the right answer

Nondeterministic FSA:
Simple machine with N states (0 to N-1)
    start in state 0
    read a bit
    depending on current state, input bit
        move to any of several new states
    stop when last bit read
    ACCEPT if in last state, REJECT otherwise

ACCEPT if there’s *some* path to the last state
REJECT if there’s *no* path to the last state

Ex:

01
0111110101
01000010110
010000

Nondeterminism doesn't help in FSAs

Given any nondeterministic FSA,
can construct a deterministic FSA
that recognizes the same language
One state in the FSA for every
set of states in the NFSA

Ex.

0	1	2	3		0	1
0	0	0	0	0		
0	0	0	1	1	4	2
0	0	1	0	2	9	2
0	0	1	1	3	15	2
0	1	0	0	4	4	1
0	1	0	1	5	4	3
0	1	1	0	6	13	3
0	1	1	1	7	15	7
1	0	0	0	8	6	8
1	0	0	1	9	6	10
1	0	1	0	10	14	10
1	0	1	1	11	15	10
1	1	0	0	12	4	9
1	1	0	1	13	6	11
1	1	1	0	14	15	11
1	1	1	1	15	15	11

Any language recognizable by a NFSA is regular
NFSA -> FSA -> RE

FSAs are equivalent to REs

Finite State Automata and Regular Expressions
are EQUALLY POWERFUL, or EQUIVALENT

THM: For any regular language, there is an FSA tha
* accepts strings in the language
* rejects strings not in the language

proof sketch:
construct a NFSA, using the following rules:

a

AB

A+B

B*

THM: For any FSA, there is a regular language that
* contains all strings accepted by the FSA
* contains no other strings

[proof omitted]

Equivalence of languages and machine models
is essential in the theory of computation

Nondeterminism does help in PDAs

Ex: nondeterministic PDA to recognize
set of all strings of the form AB
where both A and B have the
same number of 0's and 1's

Construction as for FSAs doesn't work because
stack is potentially infinite in size

Turing machines

Input on a tape that can be read *and* written

Simple machine with N states (0 to N-1)
- start in state 0
- read a bit
- depending on current state, input bit
  - write a bit, move tape R or L
  - move to a new state
- stop when last bit read
- ACCEPT if in last state, REJECT otherwise

Linear-Bounded Automata:
- length of tape = K * (size of input)

Turing Machine: no restriction on tape

Hierarchy of machines
- Finite-State Automata (-)
- Pushdown Automata (+)
- Linear Bounded Automata (?)
- Turing Machines (-)

(+) nondeterminism helps
(-) nondeterminism does not help
(?) not known whether nondeterminism helps

Each machine is more powerful than the previous
- [can recognize more languages]

Are there limits to machine "power"?