COS 126 Lecture 16: Formal languages

Language:
    a set of strings over a finite alphabet

Regular expressions
Finite-state automata
memory
nondeterminism

Given a RE, can construct a FSA that recognizes
    any string matching the RE
Given a FSA, can construct a RE that matches
    the strings recognized by the FSA
Thus, FSA's and RE's computationally equivalent

More powerful machines than FSAs:
- PDA: add pushdown stack, nondeterminism
- LBA: use (linear-bounded) tape
- Turing machine: use infinite tape

What languages do these machines recognize?

Grammars

Generate words in a formal language by a
process of replacing symbols systematically

Four elements:
- nonterminals
- terminals
- start symbol
- productions

NONTERMINAL symbols
- "local variables" for internal use
- notation: <name>

TERMINAL symbols
- set of characters that appear in the words
- "alphabet" of the language
- ex: {0,1} or ASCII

START symbol
- one particular nonterminal

PRODUCTIONS
- replacement rules
- ordered pairs of strings of symbols
- notation (ex.): a<B>c -> <D>e<F>
- LHS must have at least one nonterminal

A familiar example (abbreviated)

Nonterminal:

<sentence> <subject> <verb> <object>

Terminal:

horse dog cat saw heard the

Start:

<sentence>

Productions:

```
<sentence> -> <subject><verb><object>
<subject>  -> the horse
<subject>  -> the dog
<subject>  -> the cat
<object>   -> the horse
<object>   -> the dog
<object>   -> the cat
<verb>     -> saw
<verb>     -> heard
```

"Words" in the language:

the horse saw the dog
the dog heard the cat
the cat saw the horse
...

Could write grammar for full English language
(in principle)

Another familiar example

Nonterminal:

<stmnt> <selection-stmnt> <expression>, etc.

Terminal:

if while else switch ( ), etc.

Start:

<translation-unit>

Productions:

```
<stmnt>
   -> <selection-stmnt>
<selection-stmnt>
   -> if ( <expression> ) <stmnt>
   -> if ( <expression> ) <stmnt> else <stmnt>
<iteration-stmnt>
   -> while ( <expression> ) <stmnt>
   -> do <stmnt> while ( <expression> ) ;
```

A "Word" in the language:

```
#include <stdio.h>
main()
   { printf("hello, world\n"); }
```

Can (in fact) write grammar for full C language
(pp. 234-9 of "The C Programming Language")

Sample grammar

Nonterminal, Start: <pal>

Terminals: 0, 1

<pal> -> 0<pal>0 | 1<pal>1

<pal> -> 0 | 1

<pal> ->

DERIVATION: apply replacement rules until no nonterminals left

<pal> -> 0<pal>0 -> 01<pal>10 -> 011<pal>110 -> 0110110

PARSE TREE exhibits derivation

```
.                <pal>
.
.    0           <pal>          0
.
.        1       <pal>      1
.
.            1   <pal>  1
.
.                  0
.    0   1   1     0    1   1   0
```

Grammar GENERATES language

set of all strings that could be derived

0, 1, 00, 11, 010, 101, 000, 111, 0000, 00100 ..

Type 3 grammars

Restrict all productions to be of the form

<X> -> x

<X> -> <Y>x

Ex:

<A> -> <z>0

<z> -> <A>1

<z> ->

generates alternating sequence of 0’s and 1’s

THM: Type 3 grammars are equivalent to REs

proof sketch:

- given a Type 3 grammar, construct an FSA
  that recognizes any string in the
  language generated by the grammar
- given an FSA, construct a Type 3 grammar
  that generates the strings recognized
  by the FSA

[FSA states correspond to nonterminals]

Type 2 grammars

Restrict all productions to allow only a single nonterminal on the LHS

Ex:

<pal> -> 0<pal>0
<pal> -> 1<pal>1
<pal> -> 0
<pal> -> 1
<pal> ->

Ex:

grammar for C

Also called CONTEXT-FREE grammars

Much more descriptive than regular expressions

Q. How does the FSA have to be augmented to recognize strings from languages generated by context-free grammars?

A. Add a "memory" capability (as expected) *and* add power of nondeterminism (!)

Still limited: can’t generate strings with equal numbers of a’s, b’s, and c’s

Chomsky hierarchy

Type 1 (context-sensitive) grammars:
    add productions of the type
    <X><Y><Z> -> <X>y<Z>
Type 0 grammars: no restrictions

Essential correspondence between
    languages and automata

- Regular (Type 3)
  - finite-state machine
- Context-free (Type 2)
  - nondeterministic pushdown automata
- Context-sensitive (Type 1)
  - linear bounded automata
- Type 0
  - Turing machines

One-to-one correspondence to machines
    persists through the hierarchy
Each type is more descriptive than the previous
Each machine is more powerful than the previous

Are there limits to machine "power"?
Are there languages
    that no machine can recognize?
[stay tuned]

Lindenmayer systems

Apply productions simultaneously

Ex:

0 -> 1 [0] 1 [0] 0

1 -> 1 1

Start with 0

At stage i, apply rules to each symbol in
string form stage i-1

```
0
1 [    0    ]1 [    0    ]    0
11[1[0]1[0]0]11[1[0]1[0]0]1[0]1[0]0
```

Applications

- model cell growth in biological systems
- computer graphics

Not all formal languages fit in Chomsky hierarchy

Graftals

Convert to Lindenmayer system to 2D

1: "stem"

0: "leaf"

[] branch off

(alternate LR turns along trunk)

1   1[*]1[*]*   11[*]11[*]*   111[*]111[*]*

"Graftal" plants