CS 126 Lecture T3: Formal Languages

Outline

• Introduction
  • Defining grammar
  • Type 3 grammars
  • Type 2 grammars
  • Chomsky hierarchy
  • Conclusions
Where We Are

• T1
  - Simplest language generators: regular expressions
  - Simplest language recognizer: FSAs
• T2: more powerful machines
  - FSA, NFSA
  - PDA, NPDA
  - TM
• **T3: more powerful languages associated with the more powerful machines**
• T4:
  - Nature of most powerful machines
  - Languages that no machine can ever deal with

Review: Formal Languages

• Formal definitions
  - An **alphabet**: a finite set of symbols
  - A **string**: a finite sequence of symbols from the alphabet
  - A **language**: a (potentially infinite) set of strings over an alphabet
• Intriguing topic: **finite representation** of a language
  - How?
    + language **generators** (a set of rules for producing strings)
    + language **recognizers**
Languages and Automata

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?

Why Learn Languages?

Concrete applications:
- understanding computability
- compiler implementation
- language recognition/translation
- models of physical world
Outline

- Introduction
- **Defining grammar**
- Type 3 grammars
- Type 2 grammars
- Chomsky hierarchy
- Conclusions

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: \( \langle \text{name} \rangle \)

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

**START symbol**
- one particular nonterminal

**PRODUCTIONS**
- replacement rules
- ordered pairs of strings of symbols
- notation (ex.): \( a\langle B\rangle c \rightarrow \langle D\rangle e\langle F\rangle \)
- LHS must have at least one nonterminal
Example 1

Nonterminal:
<sentence>  <subject>  <verb>  <object>

Terminal:
horse  dog  cat  saw  heard  the

Start:
<sentence>

Example 1 (cont.)

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
Example 2

Nonterminal:

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

Terminal:

if while else switch ( ), etc.

Start:

<translation-unit>

Example 2 (cont.)

```plaintext
if (x == 0) {
  while (y == 1) {
    if (z > 0) {
      s = z * 2;
    }
  }
}
```

Each box is one production.
Example 3

Nonterminal, Start: \( \langle \text{pal} \rangle \)

Terms: \( 0, 1 \)

\[
\begin{align*}
\langle \text{pal} \rangle & \rightarrow 0 \langle \text{pal} \rangle 0 \mid 1 \langle \text{pal} \rangle 1 \\
\langle \text{pal} \rangle & \rightarrow 0 \mid 1 \\
\langle \text{pal} \rangle & \rightarrow 
\end{align*}
\]

Example 3 (cont.)

DERIVATION: apply replacement rules until no nonterminals left

\[
\begin{align*}
\langle \text{pal} \rangle & \rightarrow 0 \langle \text{pal} \rangle 0 \rightarrow 01 \langle \text{pal} \rangle 10 \\
& \rightarrow 011 \langle \text{pal} \rangle 110 \rightarrow 0110110 \\
\end{align*}
\]

PARSE TREE exhibits derivation

Grammar generates language

set of all strings that could be derived

\( 0, 1, 00, 11, 010, 101, 000, 111, 0000, 00100 \)
Outline

• Introduction
• Defining grammar
• **Type 3 grammars**
• Type 2 grammars
• Chomsky hierarchy
• Conclusions

---

**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
Type 3 Grammars and Regular Expressions

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]

Example

\[
\begin{align*}
<&A> &\rightarrow &<z>0 \\
<&z> &\rightarrow &<A>1 \\
<&z> &\rightarrow &\varepsilon
\end{align*}
\]
Demo: Construction of Equivalent FSA

Outline

• Introduction
• Defining grammar
• Type 3 grammars
• **Type 2 grammars**
• Chomsky hierarchy
• Conclusions
Context-Free 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> -> 

Also called CONTEXT-FREE grammars

Much more descriptive than regular expressions

Example Context-Free Grammar

Productions:

<stmt> 
  -> <selection-stmt> | <iteration-stmt> | ... 
<selection-stmt> 
  -> if ( <expression> ) <stmt> 
  -> if ( <expression> ) <stmt> else <stmt> 
<iteration-stmt> 
  -> while ( <expression> ) <stmt> 
  -> do <stmt> while ( <expression> ) ;

if (x == 0) { 
  while (y == 1) { 
    if (z > 0) { 
      s = z * 2;
    }
  }
}

Each box is one production.
How to Recognize Context-Free Grammars

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

Outline

• Introduction
• Defining grammar
• Type 3 grammars
• Type 2 grammars
• Chomsky hierarchy
• Conclusions
Type 1 Grammars (Context-Sensitive) and Type 0 Grammars

Type 1 (context-sensitive) grammars: add productions of the type
\[ [X][Y][Z] \rightarrow [X][Y][Z] \]
where [X] and [Z] represent any sequence of terminals and nonterminals.

Type 0 grammars: no restrictions.

Chomsky Hierarchy, Languages and Automata

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
Recursive (Type 0)
  Turing machines
Language Expressiveness and Machine Power

- 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]

Example Language Outside Chomsky Hierarchy: Lindenmayer Systems

Apply productions simultaneously

Ex:

\[
\begin{align*}
0 \rightarrow & 1 [ 0 ] 1 [ 0 ] 0 \\
1 \rightarrow & 1 1 \\
\end{align*}
\]

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
Graftal Plants: Lindenmayer in 2D

“Production” rules:
add one to each segment of my trunk;
replace each branch with myself of prev generation

(alternate LR turns along trunk)

Outline

• Introduction
• Defining grammar
• Type 3 grammars
• Type 2 grammars
• Chomsky hierarchy
• Conclusions
More on Languages, Machines, and Life

- The fine art of describing infinite beauty and variety with finite representation
  - Shakespearean sonnets and machines that can identify authorship? (Is intelligence/art nothing but a set of rules, with some non-determinism thrown in?)
  - DNA: the language of life (Is life nothing but a set of rules, with some non-determinism thrown in?)
  - ......

What We Have Learned Today

- What is a grammar?
- What is a type 3 grammar? type 2? type 1? type 0?
- How to construct an FSA from a type 3 grammar?
- What is the Chomsky Hierarchy?
- What type of automata recognize which grammars?