Formal Languages

Peter C. Chapin
Computer Information Systems
VERMONT TECH

Spring 2020

Mathematical Alphabets

Let $\mathcal{A}$ be a *finite* set of symbols.

Examples

- $\mathcal{A} = \{0, 1\}$
- $\mathcal{A} = \{a, b, c, \ldots, z\}$
- $\mathcal{A} = \{\texttt{0x00}, \ldots, \texttt{0xFF}\}$

We will leave the symbols formally uninterpreted, but the suggested meanings are clear.

Strings

Let a *string* be a *finite* sequence of symbols from some alphabet $\mathcal{A}$.

We use w_1, w_2, etc to represent strings.

Examples

- $w_1 = 0110111001$, where $\mathcal{A} = \{0, 1\}$
- $w_2 = abbacabccbbca$, where $\mathcal{A} = \{a, b, c, \ldots, z\}$

If $\mathcal{A}$ is the alphabet of byte values from 0x00 to 0xFF, then a file is a string over $\mathcal{A}$.

$\mathcal{A}^*$

Let ε represent the empty string. That is: the string with no symbols.

Let $\mathcal{A}^*$ be the set of all strings over $\mathcal{A}$, including ε.

Examples, if $\mathcal{A} = \{0, 1\}$

- $0 \in \mathcal{A}^*$
- $10010011101010110 \in \mathcal{A}^*$
- $\varepsilon \in \mathcal{A}^*$

There is also $\mathcal{A}^+ \stackrel{\text{def}}{=} \mathcal{A}^* - \{\varepsilon\}$

Note that $\mathcal{A}^*$ is an infinite set. (Proof?)

Definition of a Language

Defn: A *language* is a subset of $\mathcal{A}^*$.

Examples, if $\mathcal{A} = \{0, 1\}$

- $L_1 = \{0, 00, 11010001011\}$
- $L_2 = \{\varepsilon\}$
- $L_3 = \emptyset$
- $L_4 = \mathcal{A}^*$

Question: Is $w_1 = 110100$ in L_1? What about L_4? How can you tell?

Infinite Languages

We are mostly interested in languages with infinitely many strings. How can one specify such a language?

Use *set builder notation* (also called *set comprehensions*).

Let $N_0(w)$ be the number of zero bits in w. Similarly let $N_1(w)$ be the number of one bits.

Let $L = \{w \,|\, N_0(w) = N_1(w)\}$

Here L is the set of strings with the same number of zero and one bits.

Is 001011 in L? How can you tell? Notice that the obvious algorithm terminates because w must be finite.

General Set Builder Notation

We can use English statements to define very complex languages.

Examples

- $L_1 = \{w \mid w \text{ is a JPEG file}\}$
- $L_2 = \{w \mid w \text{ is a valid HTML document}\}$
- $L_3 = \{w \mid w \text{ is a syntactically correct C program}\}$

The problem is that it is unclear precisely what strings are in each language.

It is also unclear how one could recognize such strings.

Chomsky Hierarchy

A progression of formal languages of increasing complexity and expressiveness.

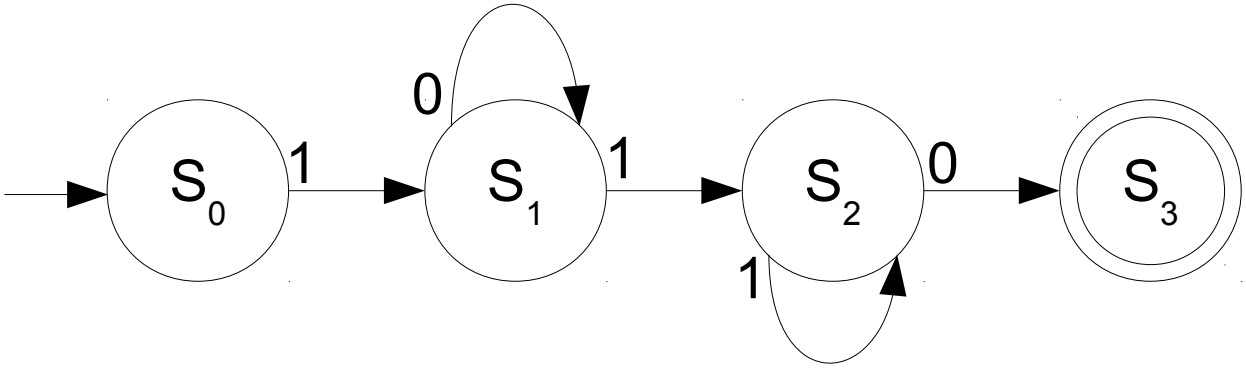
Type	*Name*	*Recognizer*
3	Regular	Deterministic Finite Automaton
2	Context Free	Pushdown Automaton
1	Context Sensitive	Linear Bounded Automaton
0	Unrestricted	Turing Machine

A *recognizer* is a machine that can determine if a given string is in a given language.

For example any regular language can be recognized by a suitable DFA.

DFA

Finite set of states; special “accept” state; transition from each state for each element of the alphabet.



Machine above recognizes the regular expression 10*11*0

Examples: $w_1 = 10000000011110$, $w_2 = 110$, $w_3 = 10111111111110$

Regular Languages $\leftrightarrow$ **Regular Expressions** $\leftrightarrow$ **DFA**

Non-Regular Languages

DFAs can't accept *arbitrary* nested structures (no ability to count)

```
begin
  begin
    begin
      X := ((((((Y + Z))))));
    end;
  end;
end;
```

Thus no regular expression can match such structures.

Need to move up the Chomsky Hierarchy!

Context Free Languages

Context free languages (CFLs) have an elegant, mathematically precise way to specify them.

- CFLs are good enough to describe the syntax of useful programming languages.
- CFLs have nice mathematical properties.
- Writing a program that recognizes a CFL is well understood.

Most real programming languages are *approximately* context free.

Context Free Grammar

Let $G = (T, N, S, R)$ where

- T is a set of *terminal symbols* (the alphabet $\mathcal{A}$ we talked about earlier).
- N is a set of *non-terminal symbols* such that $T \cap N = \emptyset$
- S is a *start symbol* with $S \in N$
- R is a set of *production rules* of the form $N \rightarrow (T \cup N)^*$

Starting with S make substitutions according to R until a string of only terminals is generated. That string is in the language defined by the grammar G.

Example: Language "Half-n-Half"

Let $T = \{0, 1\}$

$N = \{S, X, Y\}$ Where S is the start symbol

$R = (S \to X),$
$(S \to Y),$
$(X \to 0X1),$
$(Y \to 1Y0),$
$(X \to \varepsilon),$
$(Y \to \varepsilon)$

Here is a *derivation* of the string 0011...

Start with S

S	(now use $S \to X$)
X	(now use $X \to 0X1$)
$0X1$	(now use $X \to 0X1$)
$00X11$	(now use $X \to \varepsilon$)
0011	(finished; no non-terminals to expand)

Half-n-Half

Which strings are in Half-n-Half? How can you tell?

$w_1 = \varepsilon$
$w_2 = 10$
$w_3 = 101$
$w_4 = 000000000011111111111$
$w_5 = 1$
$w_6 = 110011$

Can you show derivations for the strings that are in Half-n-Half?

Are All Languages Context Free?

Not all languages are context free.

The language $L = \{w \mid N_0(w) = N_1(w)\}$ can't be generated by a CFG.

This is provable!

However, many useful languages are context free.

Who Cares?

Let $\mathcal{A} = \{$ '(', ')', '+', '=', **if**, **else**, ID, NUM $\}$

Let $w_1 =$ "ID = ID + NUM"
Let $w_2 =$ "ID = ID + **if** (NUM + ID) ID"
Let $w_3 =$ "ID ID (+ NUM) ("

Are these strings in a particular language?

Lexical Analysis

Compiler first breaks program into “tokens” (also “terminals”)

```
int main( void )
{
    printf("Hello, World!\n");
    return 0;
}
```

int ID (**void**) { ID (STRING) ; **return** NUM ; }

Is that a string in the language “C”?

Parsing

- The token stream is sent to a *parser*. If parser accepts string, it is syntacticly valid C.
- C is not a regular language! You can't write a regular expression that matches all valid C programs.
- Since C's grammar is (almost) context free, building a parser for C is a solved problem.
- Subject for a compiler design course.