–contents of Finite Automata–

–requires–

Notation Supporting Grammars
Context-free Grammars

Finite Automata

Finite automata (FA) are abstract algorithms for the recognition of sequences. They are closely related to regular expressions and CFGs.

Automata are also called state-transition machines. The central idea is that an automaton, at any one time, is in a unique state and can transition to some other state by reading input. The transitions are defined by a relation from state-input pairs to states. Transitions on the null string, i.e. non-reading transitions, are allowed. The sequence of input values read is the string that is recognized.

Processing begins in a *start state*. At each step the automaton examines the text, and based on its state-transition table, goes to another state. Each time an automaton transitions on an input symbol, that symbol is discarded so that the next symbol may be processed. Whenever the automaton is in a designated *final state*, the input read so far is said to have been *accepted*. If, on the other hand, input appears for which there is no defined transition, the automaton is said to *reject* the input. Processing continues until the input is rejected or there is no more input.

Finite automata are either deterministic (DFA) or nondeterministic (NFA). A DFA example is presented first.

State-transition Diagram

One can draw an intuitive diagram representing a DFA. The diagram below recognizes properly rounded values approximating 2/3. The value to be recognized must start with zero, then a dot, then any number of sixes and terminate with a seven. Each state is boxed; the initial state is labelled S; the final state is double boxed.

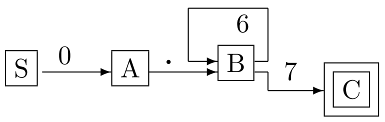

Figure 1. A DFA for rounded values of 2/3.

The same information can be represented as a *transition matrix*, with current state and next input symbol as coordinates, the successor state is at the intersection, blank means error(reject). Final states are boxed.

		see			
		0	.	6	7
	$\boxed{\text{C}}$				
in state	B			B	C
	A		B		
	S	A			

Figure 2. Transition matrix for rounded values of 2/3.

Exercises

1. Draw a diagram for a DFA that recognizes any sequence of nickels and dimes (N and D) that adds up to a quarter. (Hint: let state **k** represent an accumulation of $5k$ cents.)

2. Draw a diagram for an automaton that recognizes a sequence of zero or more a's, followed by a sequence of zero or more b's, followed by one c.

3. Draw a diagram and transition matrix for a DFA that recognizes positive integers.[1]

4. Draw a diagram or transition matrix for a DFA that recognizes rounded values of 1/7.

5. Suppose that you have a DFA that recognizes truncated representations of fraction $1/n$. How can you transform it into a DFA that recognizes rounded representations of $1/n$? (Hint: does your solution work for 1/101?).

If any entry in the transistion matrix has more than one value, or has the empty string as a value, the FA is a NFA.

CFG for Automata

A FA can also be defined by a CFG. If the rules in Π are restricted to one of the three forms shown in Table 1 the grammar is a FA. The phrase names correspond to states; the rules define transitions; the goals are the final states (such a CFG is sometimes called a *left linear grammar*).

[1]The statement of a recognition condition for an automaton implies in addition "and rejects anything else."

$$\begin{aligned} A &\rightarrow Ba && (1)\\ A &\rightarrow B && (2)\\ A &\rightarrow \lambda && (3)\end{aligned}$$

Table 1: Schema for Finite Automata

More formally a CFG,

$$\langle V_I, V_N, V_G, \Pi\rangle$$

is a FA if

$$\Pi \overset{\text{def}}{\subseteq} V_N \times \{V_N \circ V_I \cup V_N \cup \{\lambda\}\}$$

The set

$$V_S \overset{\text{def}}{=} \Pi \triangleright \{\lambda\}$$

defines the *start states*; V_G is the set of final states.

The first kind of FA rule defines the transitions (often called *shifts*). The shifts are deterministic if

$$\forall A\text{a. size}(\Pi \triangleright \{A\text{a}\}) \leq 1$$

That is to say, for no phrase name A is there more than one shift defined for any input symbol a.

The second kind of rule is called an empty transition (from B to A).[2] The third kind of rule provides a start state.

For example, the CFG for rounded values of 2/3 is:

```
C = B '7';
B = B '6';
B = A '.';
A = S '0';
S = ;
```

CFG Form of a DFA Describing 2/3

If the shifts are deterministic and there are no empty transitions, and there is only one start state, then the automaton is a DFA; otherwise it is an NFA.

If one uses a FA CFG to reduce a string, one starts with the input text. The only reductions that can be applied are those for start states. At each subsequent stage the FA transitions according to one of the other rules for 2/3 until the FA stops. The derivation of the S/R sequence using the $\mathcal{C}$ predicate (from ContextFreeGrammar) follows. As usual it is read bottom to top (reducing).

[2]An empty transition is often called an ϵ transition in literature using letter ϵ to denote the empty string.

$\mathcal{C}$(C, λ)	reduce
$\mathcal{C}$(B7, λ)	shift
$\mathcal{C}$(B, 7)	reduce
$\mathcal{C}$(B6, 7)	shift
$\mathcal{C}$(B, 67)	reduce
$\mathcal{C}$(B6, 67)	shift
$\mathcal{C}$(B, 667)	reduce
$\mathcal{C}$(A., 667)	shift
$\mathcal{C}$(A, .667)	reduce
$\mathcal{C}$(S0, .667)	shift
$\mathcal{C}$(S, 0.667)	reduce
$\mathcal{C}$(λ,0.667)	start

CFG proof of $0.667 \in \mathcal{L}(2/3)$

Exercises

6. Write down the grammars defining the automata derived in the previous set of exercises.

7. Write a program to execute the DFA in Figure 1.

8. Write a program to execute an arbitrary DFA. (Hint: represent Π as a transition matrix.)

9. Write a program to execute an arbitrary NFA. (Hint: use threading or backtracking.)

10. Show how to derive a grammar $\mathcal{A}'$ for the *sequence of states* passed to accept a string from the grammar $\mathcal{A}$ defining the FA. (Hint $V_I' = V_N$).

NFA to DFA

The surprising fact is that any (bad) NFA can be systematically transformed into a (good) DFA.

One can transform an NFA into a DFA a little bit at a time. The idea is to use CFG transformations, each removing some non-determinancy, while preserving the language.

If a NFA has a rule with an empty right-hand side (which is forbidden in a DFA), one can **remove** the rule. Any immediate circularity can be removed whenever it occurs or is introduced. Here is an example with A as the start state and C as a final state. The problem is an empty transition from A to B.

```
C = B '1';
B = A;
B = A '2';
A = ;
```

Following the removal of the ”bad” rule and adding a new rule substituting A where B was used:

```
C = B '1';
C = A '1';
B = A '2';
A = ;
```

This new CFG is a DFA. It can be used to parse the string ’21’ as follows:

```
𝒞(C, λ)
𝒞(B1, λ)
𝒞(B, 1)
𝒞(A2, 1)
𝒞(A, 21)
𝒞(λ, 21)
```

Here is an example with both kinds of non-determinancy. The CFG has an erasing rule and also state A has two different transitions on 1. The language is {101, 110, 111}.

```
F = D '0';
F = E '1';
E = D;
E = B '0';
D = C '1';
C = A '1';
B = A '1';
A = ;
```

After eliminating the empty transition (as before), we have the transformed CFG:

```
F = D '0';
F = E '1';
F = D '1';
E = B '0';
D = C '1';
C = A '1';
B = A '1';
A = ;
```

After fusing the two ambiguous rules, adding a new symbol X, and using X where C and B had been used, we have the transformed CFG for the DFA:

```
F = D '0';
```

```
F = D '1';
F = E '1';
E = X '0';
D = X '1';
X = A '1';
A = ;
```

The corresponding transition diagram is given in Figure 3.

		see	
		0	1
	$\boxed{\text{F}}$		
	E		F
in	D	F	F
state	X	E	D
	A		X

Figure 3. Transition matrix for constructed DFA.

Big Bang NFA to DFA transformation

There is a "big-bang" algorithm to do the NFA-DFA transformation all in one step. See, for example, Wikipedia on "Powerset construction." The underlying concept is collecting all states with the same effect into sets, then defining a new machine based on sets of the original states.

The algorithm can be expressed in terms of the NFA CFG. Suppose $\mathcal{G}$ is the NFA and $\mathrm{A} \in V_N, \mathrm{B} \in V_N, \mathrm{b} \in V_I$. The sought-after DFA $\mathcal{G}'$ is recursively defined as follows:

$$\begin{array}{rcl}
\mathcal{M}(\mathrm{B},\mathrm{b}) & \stackrel{\text{def}}{=} & \{\mathrm{A} \mid \mathrm{A} \leftarrow^* \mathrm{Bb}\} \\
\mathcal{E} & \stackrel{\text{def}}{=} & \{\mathrm{A} \mid \mathrm{A} \leftarrow^* \lambda\} \\
V_I' & \stackrel{\text{def}}{=} & V_I \\
V_N' & \stackrel{\text{def}}{=} & \{\mathcal{E}\} \cup \{\mathcal{M}(\mathrm{B},\mathrm{b})\} \\
V_G' & \stackrel{\text{def}}{=} & \{V_G\} \\
\Pi' & \stackrel{\text{def}}{=} & \{\mathcal{E} \leftarrow \lambda\} \\
 & \cup & \{\mathrm{A}' \leftarrow \mathrm{B}'\mathrm{b} \mid \mathrm{B}' \in \mathcal{D}(\Pi') \wedge \mathrm{A}' = \mathcal{M}(\mathrm{B},\mathrm{b}) \wedge \mathrm{B} \in \mathrm{B}'\}
\end{array}$$