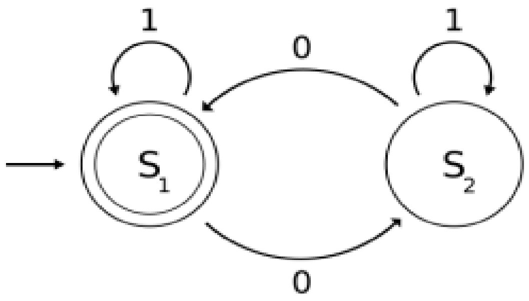
Automaton Theory Applications

Deterministic Finite Automaton (DFA) as a Model of Computation



A DFA requires O(1) memory, regardless of the length of the input.

- ❑ If a function grows in memory O(1) then it uses a constant amount of memory regardless of the input size.

In other words:

- ❑ «Big O notation» is a way to express the speed of algorithms.
  - ❖ n is the amount of data the algorithm is working with.

O(1) means that no matter how much data, it will execute in constant time.

O(n) means that execution is proportional to the amount of data.

DFAs as a Model of Computation

- ❑Regular Languages describe what is possible to do with a computer with very little memory.
  - ❖No matter how long the input is, a DFA only keeps track of what state it’s currently in
    - ✓so it only requires a constant amount of memory.
- ❑By studying properties of regular languages, it is possible to understand:

  what is and what isn’t possible with computers with very little memory.
- ❑This is the reason of the usage of regular languages

DFAs and Regular Expressions

- Regular expressions are a useful tool that every programmer should know.
- If we want to check if a string is a valid email address, we may write something like:

```
/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
```

- This regular expression gets converted into an NFA (nondeterministic finite automata) ,
  - NFA can be quickly evaluated to produce an answer.
- We don’t need to understand the internals of this in order to use regular expressions

What is regular expression in HTML?

RegExp Object.

- ❑A regular expression is an object that describes a pattern of characters.
- ❑ Regular expressions are used to perform pattern-matching and "search-and-replace" functions on text.

RegExp Object

Expression	Description
[abc]	Find any character between the brackets
[^abc]	Find any character NOT between the brackets
[0-9]	Find any character between the brackets (any digit)
[^0-9]	Find any character NOT between the brackets (any non-digit)
(x|y)	Find any of the alternatives specifies

Brackets are used to find a range of characters

RegExp Object

Metacharacters are
characters
with a special meaning

Metacharacter	Description
.	Find a single character, except newline or line terminator
\w	Find a word character
\W	Find a non-word character
\d	Find a digit
\D	Find a non-digit character
\s	Find a whitespace character
\S	Find a non-whitespace character
\b	Find a match at the beginning/end of a word
\B	Find a match not at the beginning/end of a word
\0	Find a NUL character
\n	Find a new line character
\f	Find a form feed character
\r	Find a carriage return character
\t	Find a tab character
\v	Find a vertical tab character
\xxx	Find the character specified by an octal number xxx
\xdd	Find the character specified by a hexadecimal number dd
\uxxxx	Find the Unicode character specified by a hexadecimal number xxxx

RegExp Object

Quantifiers

Quantifier	Description
n+	Matches any string that contains at least one *n*
n*	Matches any string that contains zero or more occurrences of *n*
n?	Matches any string that contains zero or one occurrences of *n*
n{X}	Matches any string that contains a sequence of *X n*'s
n{X,Y}	Matches any string that contains a sequence of X to Y *n*'s
n{X,}	Matches any string that contains a sequence of at least X *n*'s
n$	Matches any string with *n* at the end of it
^n	Matches any string with *n* at the beginning of it
?=n	Matches any string that is followed by a specific string *n*
?!n	Matches any string that is not followed by a specific string *n*

DFAs in Compilers

- In every programming language, the first step in the compiler or interpreter is the lexer.
  - The lexer reads in a file of favorite programming language
  - The lexer produces a sequence of tokens
- For example, the code line in C++:

```
cout << "Hello World" << endl;
```

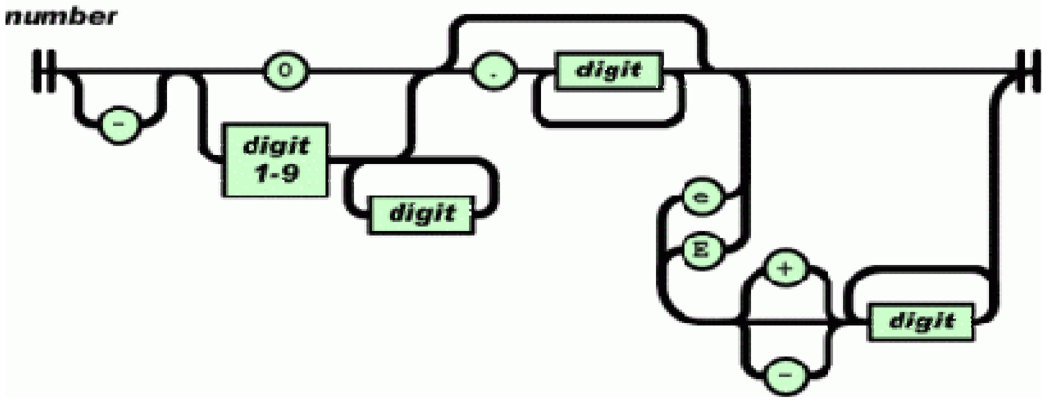
The lexer **generates** something like this:

```
1   IDENTIFIER  cout
2    LSHIFT      <<
3   STRING      "Hello World"
4    LSHIFT      <<
5    IDENTIFIER  endl
6    SEMICOLON   ;
```

- The lexer uses a DFA to go through the source file, one character at a time, and emit tokens

DFAs in Compilers

- ❑If an programming language is designed , this will be one of the first things   it is written.



Lexer description for JSON numbers, like -2.34

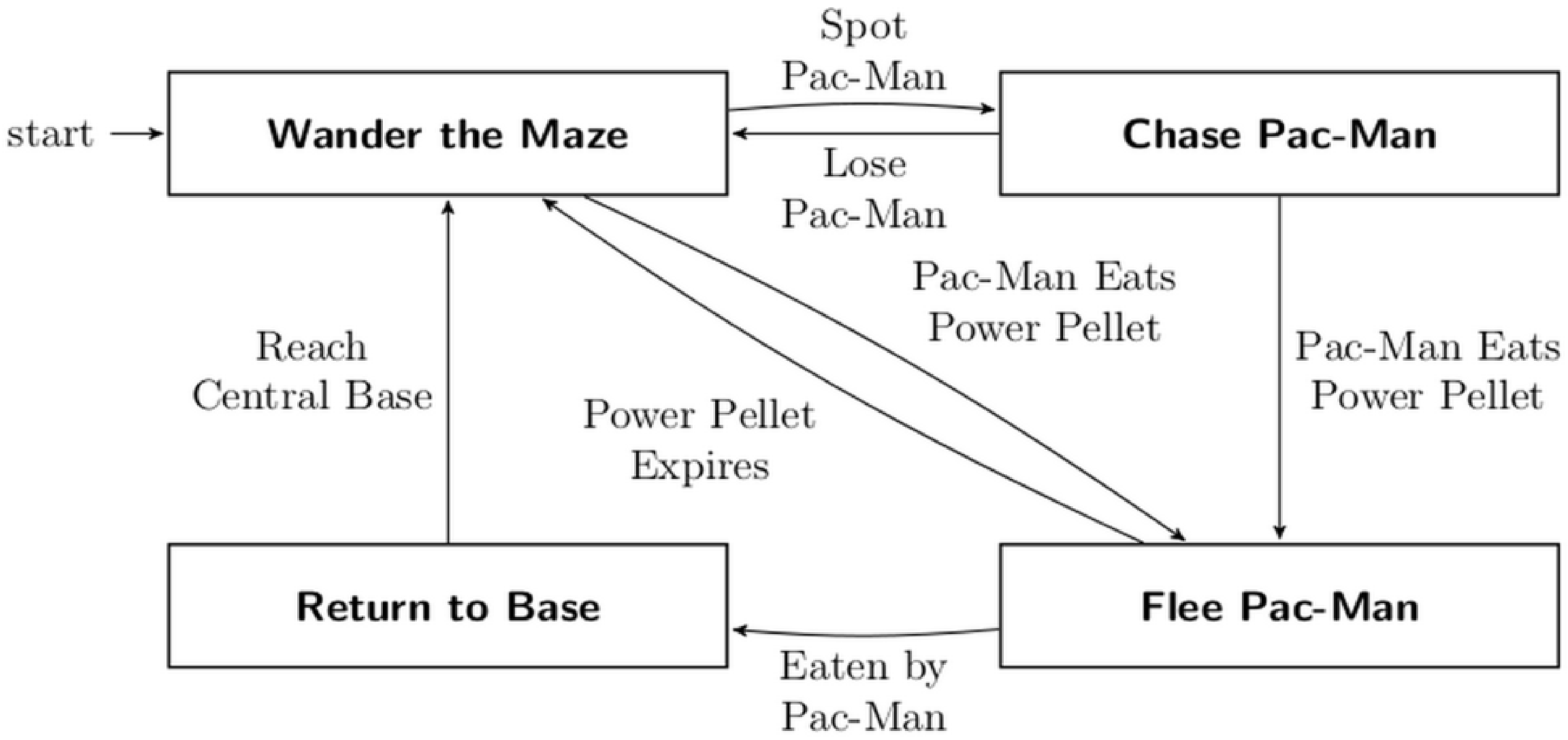
DFAs for Artificial Intelligence

- ❑Another application of finite automata is programming simple agents to respond to inputs and produce actions in some way.
- ❑We can write a full program.

  BUT

  - ❖A DFA is often enough to do the job.
  - ❖DFAs are also easier to reason about and easier to implement.

The AI for Pac-Man uses a Four-State Automaton

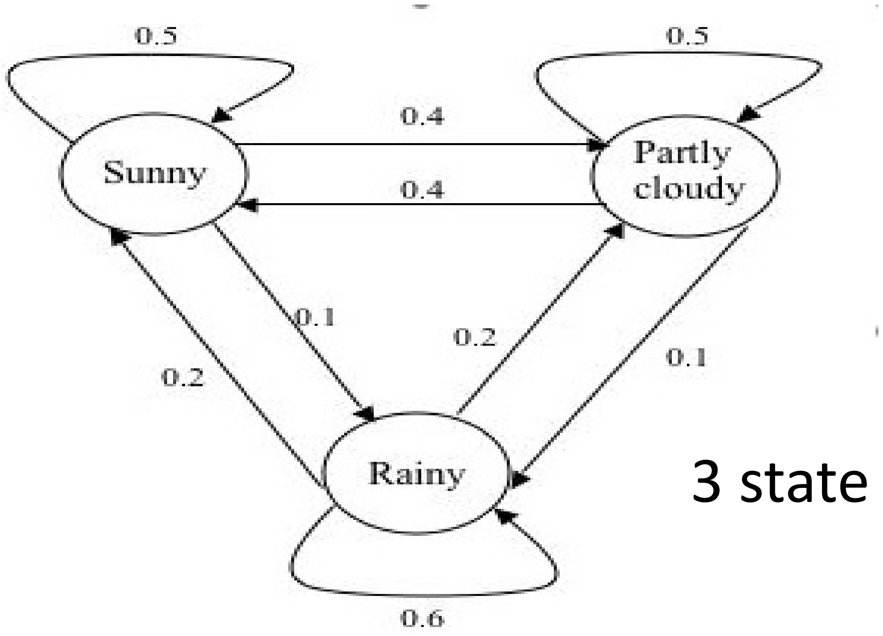


- ❑This type of automaton is called a Finite State Machine (FSM) rather than a DFA.
- ❑In a FSM, an action is done depending on the state
  - ❖In a DFA, accepting or rejecting a string is done.
    - ✓ BUT both of them are the **same concept.**

DFAs in probability

❑Instead of fixed transition rules of DFA , the transitions are probabilistic.

❖This is called a Markov Chain



3 state Markov chain to model the weather

Markov Chains

- ❑Markov chains are frequently used in probability and statistics
- ❑Markov chains have lots of applications in finance and computer science.
- ❑ Google's PageRank algorithm uses a Markov chain to determine the relative importance of web pages
- ❑It is possible to calculate things like the probability of being in a state after a certain number of time steps, or the expected number of steps to reach a certain state.