Lexical Analysis

Break a program up into "tokens" - these tokens can usually be described using deterministic finite automata

```plaintext
pay = salary + ( ovtimerate * overtime ) ;
```

Deterministic Finite Automata

Can be used a model for what happens during lexical analysis — scan program from beginning to end and divide it into tokens. Finite automata are used to specify tokens of programming languages. Also used in “model checking”, reasoning about systems with objective of proving they satisfy useful properties. Also used in statistical models for analysing biological and textual sequences.

Deterministic Finite Automata

Simple mechanism for scanning a string

![Finite automaton](image)

**Figure: finite automaton**

Gives method of recognising words belonging to certain languages (must accept or fail to accept at end of string)

- any finite set of strings
- various infinite sets of strings, e.g.
  - strings having exactly 2 occurrences of the letter `a`
  - strings having more than 6 letters
  - strings in which letter `b` never comes before letter `a`
Furthermore we shall see that finite automata cannot be used to describe certain languages such as

- the set of strings containing more $a$’s than $b$’s
- all words that remain the same if you read them back to front
- well-formed arithmetic expressions, if there is no limit on nesting of parentheses.

A deterministic finite automaton (DFA) has 5 components:

1. $Q$ is a finite nonempty set whose members are called states of the automaton;
2. $A$ is a finite nonempty set called the alphabet of the automaton;
3. $\phi$ is a map from $Q \times A$ to $Q$ called the transition function of the automaton;
4. $i$ is a member of $Q$ and is called the initial state;
5. $T$ is a nonempty subset of $Q$ whose members are called terminal states or accepting states.

“quintuple” — any DFA can be divided into these 5 components

state of a machine tells you something about the prefix that has been read so far. If the string is a member of the language of interest, the state reached when the whole string has been scanned will be an accepting state (a member of $T$).

There is only one empty string so there is only one initial state (denoted $i$)

Transition function $\phi$ tells you how state should change when an additional letter is read by the DFA

A DFA is often depicted as a labelled directed graph. (called transition diagram)

3 states, $i$, $r$ and $t$. Accepting state $t$ has outgoing arrow.
Symbolic description of the example DFA

Automaton $A = (Q, A, \phi, i, T)$

Set of states $Q = \{i, t, r\}$, $A = \{0, 1\}$, $T = \{t\}$ and the transition function $\phi$ is given by

$\phi(i, 0) = r$, $\phi(i, 1) = t$,
$\phi(t, 0) = t$, $\phi(t, 1) = t$,
$\phi(r, 0) = r$, $\phi(r, 1) = r$.

It is simpler to describe a transition function by a table of values. In this example we have:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>r</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>r</td>
<td>t</td>
<td>r</td>
</tr>
</tbody>
</table>

A simplification

If $\phi$ is a partial function (not defined for some state/letter pairs), then the DFA rejects an input if it ever encounters such a pair. This convention often simplifies the definition of a DFA. In the previous example we could use transition table:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>t</td>
<td>t</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
</tr>
</tbody>
</table>

Alternative notation for accepting state is 2 concentric circles.
Example

Give a DFA that accepts the words “cat” and “dog” (using the 6-letter alphabet $a, c, d, g, o, t$).

An important observation

Any DFA that uses the convention that an undefined transition leads to a rejection, can be converted to a DFA that uses a total transition function (that is, one that is defined for all combinations of input symbols and states).

The convention is useful, but it does not add extra expressive power.

Example

The DFA can be “stripped down” if we understand an undefined transition to mean: reject the whole string.

Exercises

Draw a diagram for a DFA that only accepts the words: fun, fair, unfair, funfair

Note. this language can be represented by simple enumeration: in set-theoretic notation

$$\{\text{fun, fair, unfair, funfair}\}$$

How about the following infinite language. Can you give a DFA that accepts the words: bad, baaad, baaaaad, ...?
Any finite language is accepted by some DFA

General rule: \( i \) is the initial state. For each prefix \( p \) of a word in the list, include state \( s_p \) with the idea that the machine should be in state \( s_p \) after \( p \) has been read.
If \( p \) is equal to one of the given words, make \( s_p \) an accepting state.
For prefixes \( p \) and \( q \) with \( q = pa \ (a \in A) \), let \( \phi(s_p, a) = s_q \)
If \( pa \) is not a prefix of any word, \( \phi(s_p, a) = f \) (failure state)
For all \( a \in A, \phi(i, a) = s_a \).
Try using this rule to construct a DFA that accepts
\( a, \ cat, \ cats, \ dog, \ deer \)

Example

Construct a DFA that accepts words over the alphabet \{a, b\}
which contain an odd number of a’s and an even number of b’s.
State should keep track of parity of number of occurrences of each
letter seen so far.
So: this suggests using 4 states, called \( E/E, \ E/O, \ O/E, \ O/O \)
where \( i = E/E \).

\[
\begin{align*}
\phi(E/E, a) &= O/E \\
\phi(E/E, b) &= E/O \\
\end{align*}
\]

etc.
\( T = \{O/E\} \)

General observations

Any finite language (and various infinite languages) have DFAs
that recognise them.
Question: What sort of languages can be recognised by DFAs?
Note: although finite languages can be enumerated, it may be
advantageous to describe them using a DFA (e.g. words of length 10)

More examples

Write down DFAs which recognise the following languages over the alphabet \{a, b\}:
\( L_1 \) is set of all words containing exactly three occurrences of a
\( L_2 \) is set of all words containing at least three occurrences of a
\( L_3 \) is set of words containing the substring aaa
More on the textbook definition

Initially the state is $i$ and if the input word is $w = a_1 a_2 \ldots a_n$ then, as each letter is read, the state changes and we get $q_1, q_2, \ldots, q_n$ defined by

\[
\begin{align*}
q_1 & = \phi(i, a_1) \\
q_2 & = \phi(q_1, a_2) \\
q_3 & = \phi(q_2, a_3) \\
& \vdots \\
q_n & = \phi(q_{n-1}, a_n)
\end{align*}
\]

More on the textbook definition

Extend the definition of the transition function so that it tells us which state we reach after a word (not just a single letter) has been scanned:

In the above notation, extend the map $\phi : Q \times A \to Q$ to $\phi : Q \times A^* \to Q$ by defining:

\[
\begin{align*}
\phi(q, \epsilon) & = q & \text{for all } q \in Q \\
\phi(q, wa) & = \phi(\phi(q, w), a) & \text{for all } q \in Q; \ w \in A^*; \ a \in A.
\end{align*}
\]

It is easy to show by induction that this extended map satisfies

\[
\phi(q, vw) = \phi(\phi(q, v), w) \text{ for all } q \in Q; \ v, w \in A^*.
\]

More on the textbook definition

Language defined by a DFA

Suppose we have a DFA $A$.

A word $w \in A^*$ is said to be accepted or recognised by $A$ if $\phi(i, w) \in T$, otherwise it is said to be rejected. The set of all words accepted by $A$ is called the language accepted by $A$ and will be denoted by $L(A)$. Thus

\[
L(A) = \{ w \in A^* : \phi(i, w) \in T \}.
\]