1 Excursion: Practical applications of finite automata

1.1 Pattern matching

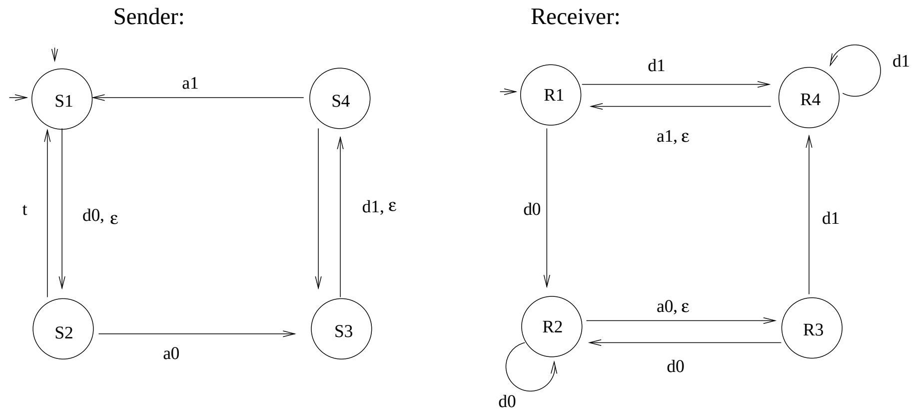
We can use automata for pattern matching in two ways: either construct an automaton for the string to be found (as noticed before), or construct a *suffix automaton* for the whole text. Such suffix automaton functions as an index for the text and can be used to find any string in the text.

1.2 The message-passing protocols as finite automata

Message-passing protocols can also be descibed as finite automata. Let's consider a very simple AB (alternating bit) protocol, in which the sender process S and receiver process R communicate through a *half-duplex channel*. In the half-duplex channel messages can be sent only to one direction at the same time. The channel can corrupt or loose the messages, but the order of messages remains same and the channel doesn't duplicate messages.

Because there is always only one message in the channel, it is enough to use two bits, 0 and 1, for representing messages. The sender tries first to send message $d0$ and if it gets an acknowledgement $a0$ for it, it takes the next message, marked $d1$, and sends it. If it gets an acknowledgement $a1$ for $d1$, it can again send a message, which is now marked as $d0$, and so on. The sender cannot send a new message, before it gets the acknowledgement for the previous one. If the acknowledgement doesn't arrive during some time limit t, it sends the message again. The receiver reads all messages, which arrive through the channel and sends an acknowledgement for each of them, possibly several times (if the same message arrives several times).

The sender and receiver processes can be described as the following undeterministic (ϵ-) automata.



1.3 Lexical analysis

Lexical analyzer is part of compiler, which divides the source code into tokens (logically connected expressions). E.g. UNIX-commands `lex` and `flex` can be used to generate a lexical analyzer for the given language. The commands get as input the regular expressions defining tokens, and the rules, what to do when the string matches some rule.

The input could be:

else	{return(ELSE); }
[A-Za-z][A-Za-z0-9]*	{code to enter the found identifier in the symbol table; return(ID); }
>=	{return(GE);}
=	{return(EQ); }
...	

For further reading, look at Hopcroft, J.E., Motwani, R., Ulman (new edition).

1.4 Preprocessing natural language

Transducers, Rewriting automata, are useful tool in natural language preprosessing.

E.g. transducer, which recognices noun phrases of form $[(d)a * n+]$, in which d=determiner, a=adjective, n=noun:

For further reading look at
`http://www.xrce.xerox.com/competencies/content-analysis/fsCompiler/ fsexamples.html`.

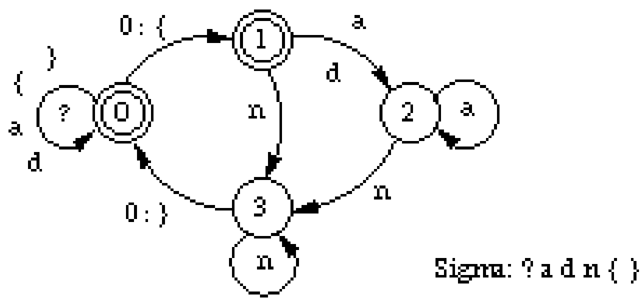


Figure 1: Transducer, whics writes paranthesis around all noun phrase, it recognizes.