

Reprise on Languages and Grammars

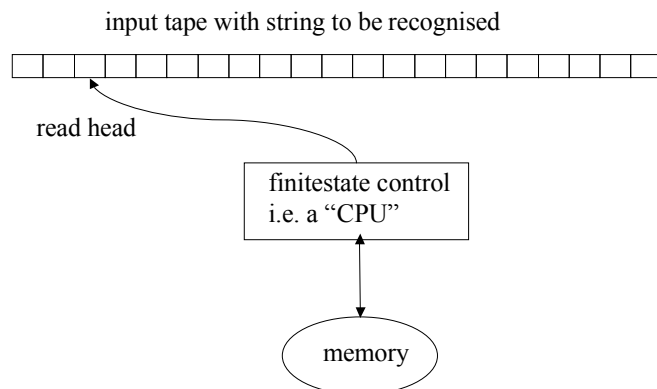
- A *language* is a set (usually infinite) of strings, also known as *sentences*
- Each string consists of a sequence of symbols taken from some *alphabet*
- You cannot define a language by listing the strings
- You can define a language by a *grammar* — a finite set of rules for generating the strings
- A grammar, G , is a quadruple (V_N, V_T, P, S) where
 - V_N is a set of symbols called *nonterminals*
 - V_T is a set of symbols called *Terminals*, the "words" of the language
 - P is a set of *Productions* – the rules for the formation of sentences
 - $S \in V_N$ is the *Start symbol* – the starting point.

Different kinds of Grammars

- Grammars have a set of *productions*
- Most general form is
$$\alpha \rightarrow \beta$$
- Where α is a string including a non-terminal and β is any string (of terminals and non-terminals)
- Different kinds of grammars constrain the productions:
 - TYPE 1: $|\alpha| \leq |\beta|$
 - TYPE 2: α consists only of a non-terminal
 - TYPE 3: α consists only of a non-terminal and β consists only of a terminal or a terminal followed by a non-terminal

Languages and Machines

- We can also define a language by specifying a machine that will recognise sentences of the language.
- The general structure of these machines is:

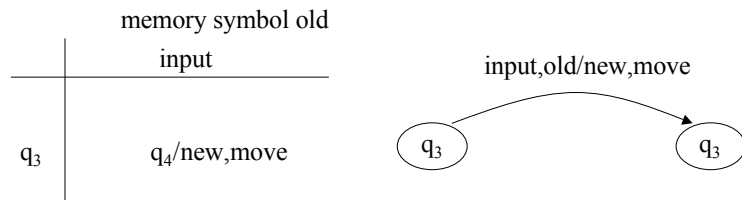


Machine structure

- The input tape is divided into squares — one per terminal symbol of the string to be recognised
- The finite state control has a read head, initially positioned at the first input symbol
- Every time an input symbol is processed, the read head moves to the next symbol
- The auxiliary memory can be read and written in the process of recognising the sentence
 - normally a linear structure, such as a tape, a stack, etc. with one symbol accessible at a time
- The finite state control has a number of possible states including an *initial state* q_0 and a set of possible *final* or *goal states*

Machine structure

- The behaviour of the machine is defined by a mapping *from* the current state, (possibly) the current input symbol (under the read head) and the current memory symbol *to* the next state, the new memory symbol(s) and (possibly) a movement of the memory access point
- These state transitions can be specified in a table or in a diagram:



Acceptance of strings

- The machine starts in the initial state with the read head over the first input symbol and the auxiliary memory empty (or some predetermined contents)
- The aim is to finish in one of the goal states with the input exhausted, in which case the string is accepted (as a sentence)
- Otherwise, the string is *not* accepted

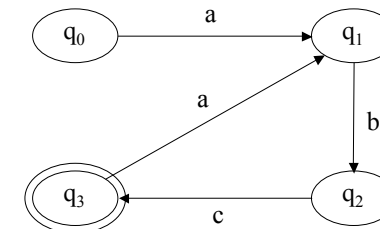
Different kinds of machines

- The kind of auxiliary memory determines the kinds of languages that can be recognised:

Aux memory	Kind of machine	Kind of grammar
None	Finite state aut.	TYPE3
Stack	Push down aut.	TYPE2
Tape bounded by input length	Linear bounded aut.	TYPE1
Unbounded tape	Turing machine	TYPE0

Example of Finite State Automaton

- Consider recognising strings $(abc)^n$ where $n \geq 1$
—final states are indicated by a double outline



- We could explicitly add error handling:
 - add an error state
 - have a transition from any state to the error state on unexpected input

Example of Finite State Automaton

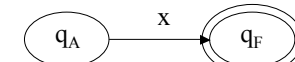
- We can demonstrate acceptance of the string abcabc with a sequence of *configurations*, each of which indicates the current state, the unprocessed input and the auxiliary memory (none in this case):

$(q_0, abcabc)$ |— $(q_1, bcabc)$
 |— $(q_2, cabc)$
 |— (q_3, abc)
 |— (q_1, bc)
 |— (q_2, c)
 |— $(q_3,)$

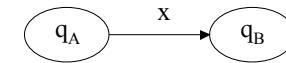
Accept!

Key result for Finite State Automata

- The class of languages that can be recognised by finite state automata is the same as the class of languages that can be generated by TYPE3 grammars
- This can be proved/demonstrated by showing how to map between the two kinds of specifications
- Consider a TYPE3 grammar with start symbol S:
 - introduce an initial state q_S and a final state q_F
 - for each (other) non-terminal A, introduce a state q_A
 - for productions $A \rightarrow x$ (where x is a terminal), add a transition:



- for productions $A \rightarrow xB$, add a transition:

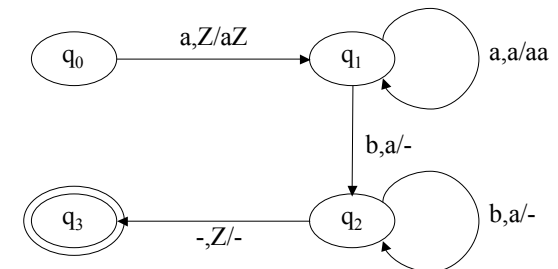


Example

- Consider the earlier example of the language consisting of strings $(abc)^n$ where $n \geq 1$
- The specified machine (with some minor relabelling) corresponds to the grammar:
 - $\{S, A, B, C\}, \{a, b, c\}, P, S$
- where P consists of the productions:
 - $S \rightarrow aB$
 - $B \rightarrow bC$
 - $C \rightarrow c$
 - $C \rightarrow cA$
 - $A \rightarrow aB$
- The relabelling would be:
 - $q_0 = q_S, q_1 = q_B, q_2 = q_C, q_3 = q_A$ and q_F

Example of Push Down Automaton

- Consider recognising strings $a^n b^n$ where $n \geq 1$



- Transitions are encoded:
 - input, top of stack / replacement for top of stack
- Stack initially has Z on it

Example of Push Down Automaton

- We can demonstrate acceptance of the string aaabbb with a sequence of *configurations*, each of which indicates the current state, the unprocessed input and the stack contents (top of stack on the left):

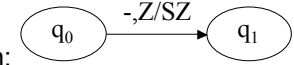
$(q_0, aaabbb, Z)$ |— $(q_1, aabbb, aZ)$
 |— $(q_1, abbb, aaZ)$
 |— $(q_1, bbb, aaaZ)$
 |— (q_2, bb, aaZ)
 |— (q_2, b, aZ)
 |— $(q_2, -, Z)$
 |— $(q_3, -, -)$

Accept!

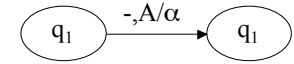
Key result for Push Down Automata

- The class of languages that can be recognised by push down automata is the same as the class of languages that can be generated by TYPE2 grammars
- This can be proved/demonstrated by showing how to map between the two kinds of specifications
- Consider a TYPE2 grammar with start symbol S:

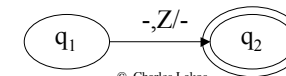
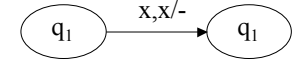
- introduce an initial state q_0 , a processing state q_1 and a final state q_2
- for start symbol S, add a transition
- for productions $A \rightarrow \alpha$, add a transition:



- for terminals x, add a transition:

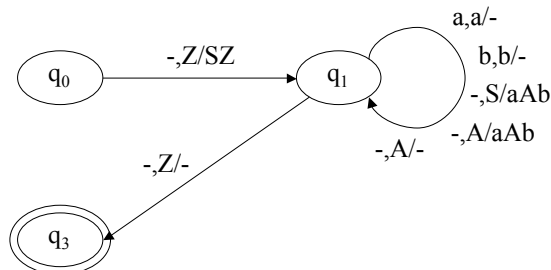


- add a final transition:



Example

- Consider the earlier example of the language $a^n b^n$ where $n \geq 1$
- Consider the grammar:
 - $\{S, A\}, \{a, b\}, P, S$
- where P consists of the productions:
 - $S \rightarrow aAb$ $A \rightarrow \epsilon$ $A \rightarrow aAb$
- The push down automaton would be:



5 labels!
5 arcs!

Summary Notes

- The lecture has considered different kinds of machines and their relationship to different kinds of grammars
- The key issue is to understand this relationship
 - TYPE3 grammars which are used to define tokens in languages correspond to the simplest kind of machine and form the basis of lexical analysis
 - TYPE2 grammars are used to define the syntax of programming languages correspond to machines with a stack and form the basis of syntactic analysis
- We have glossed over a number of issues
 - transforming machines to grammars
 - dealing with non-deterministic machines