Introduction To Formal Languages

Teacher Notes to support Tenderfoot Unit 5: Theoretical Computers – Fun with finite-state machines

Short exercises and an outdoor activity explore the use of finite-state machines to define language checkers. Introduces an awareness of language structure and the notion of context free grammars and language translation.

Preparation required:
'I Have A Spelling Checker', Eating Your Own Words instructions for all students.
FSA state diagram drawn in playground, paper extension exercise if required.
Treasure Hunt materials if undertaking that activity.

Eye Halve a Spelling Chequer

Eye halve a spelling chequer
It came with my pea sea
It plainly marques four my revue
Miss steaks eye kin knot sea.

Eye strike a quay and type a word
And weight four it two say
Weather eye am wrong oar write
It shows me strait a weigh.

As soon as a mist ache is maid
It nose bee fore two long
And eye can put the error rite
Its really ever wrong.

Eye have run this poem threw it
I am shore your pleased two no
Its letter perfect in it's weigh
My chequer tolled me sew.

(Sauce unknown)

The Structure Of Language

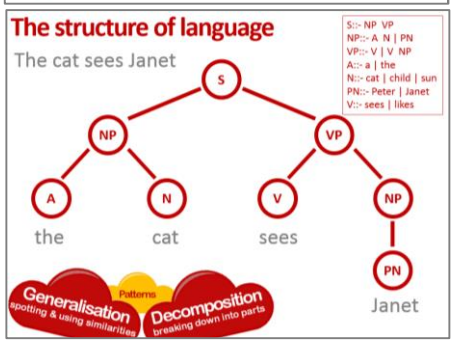
Have you ever wondered how a spell check works … and why it sometimes doesn't! The poem written to highlight their limitations was written by Professor Jerrold H. Zar, at Northern Illinois University, in 1992. The original and a simpler version (shown) are included. Deriving the correct meaning is a good introductory exercise. Homophones (words that sound the same, but are spelt differently) highlight the problem. Natural languages, such as English are full of ambiguity and there is more to them than just spelling and grammatical rules. Meaning is often dependent on context, which is why online translation services are notorious.

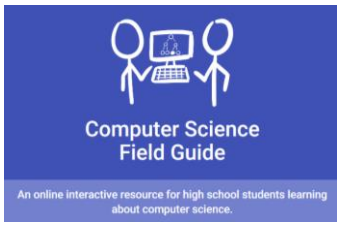
Natural languages evolve. Formal languages, by contrast are designed. Programming languages adhere to strictly defined rules. This makes it possible to accurately translate them into the machine code. If the syntax and grammar aren't perfect, the translator rejects them but there is no contextual ambiguity. Linguistics is an important area as computer science strives, not only to define more intuitive programming languages, but translation and grammar checking services too. CS4Fn have a child friendly introduction: goo.gl/B47YMv.

In the 1950's, Noam Chomsky laid the foundations for analysing languages. He defined a language as a sequence of 'atoms'. Taking English as an example, a sentence is made up of a sequence of words. Words are made up of a sequence or string of letters. The letters are the languages' alphabet – from which more complex sequences can be built. Not all possible sequences are valid words.

Grammatical rules define how the atoms can be combined and structured. In this example a sentence is defined as a combination of a Noun Phrase 'symbol' followed by a Verb Phrase 'symbol'. All subsequent symbols are defined until we get down to actual words in the language. The box showing the symbolic representation (note the pipe to represent OR) is known as Backus-Naur Form. Developed by John Backus and Peter Naur to define Algol, and is now the main technique for defining formal languages. To evaluate whether an expression is valid, a Parse Tree can break it down into defined symbols. This is known as a derivation. Further detail and examples are given in the slides / notes. Essentially it is an exercise in decomposition and pattern recognition.

Sentence S	→	a Noun Phrase (NP) followed by a Verb Phrase (VP)
Noun Phrase NP	→	an Article (A) followed by a Noun (N) OR Proper Noun (PN)
Verb Phrase VP	→	a Verb (V) OR a Verb followed by Noun Phrase
Article A	→	'a' OR 'the'
Noun N	→	'cat' OR 'child' OR 'sun'
Proper Noun PN	→	'Peter' OR 'Janet'
Verb V	→	'sees' OR 'likes'

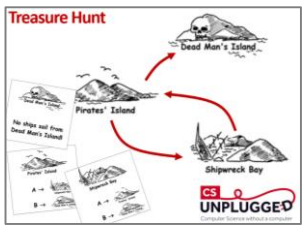




Languages can be very complex, illustrated by the Java example in the presentation. It has hundreds of rules but compilers use them to dissect statements into symbols, which decompose into defined atoms. There is a good introductory article at goo.gl/B8kR0p . The Computer Science Field Guide (csfieldguide.org.nz) chapter on Formal Languages is well worth reading to appreciate the depth of this area.

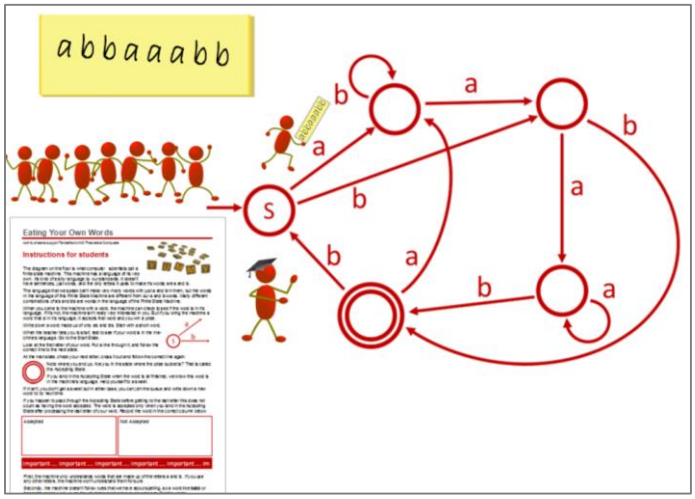
Finite-State Machines

Computer Science will be unfamiliar territory for many ICT teachers. Computerphile, a YouTube channel, has 250+ short videos on all aspects of computing topics. youtu.be/vhiiia1_hC4 is an excellent (8 min) explanation of link between finite-state machines, languages and computer science. There are several related videos. If you haven't familiarised students with the idea of a finite-state machine, the CSUnplugged Treasure Hunt activity described in the presentation is good introduction. As the main activity is outdoors, this makes a good starter. A 2 minute video: csunplugged.org/videos/#Finite_State_Automata explains it well. Such machines are known as acceptors or finite-state automata (FSA) since the output is indicated by where you finish.



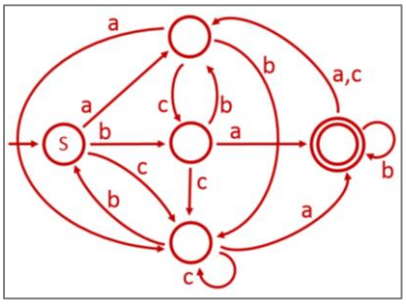
Eating Your Own Words

Students act out the operation of a FSA drawn in the playground. They create 'tickets' – a string of letters that serve as inputs. If they end up in an accepting state they receive a sweet - their ticket was recognized by the machine as a word in its language. An easier example is supplied but the one illustrated is a good starting point for KS3 students. The states should be large enough to stand in. Place a container with lots of small sweets in the accepting state (the double circle). Ensure all children can see what is happening.



Give out the instructions. The machine understands its own language, not ours. Made up of just two letters, a and b, it may seem a bit silly at first. There are lots of words in its language, but not all combinations of 'a' and 'b' are words it recognises. Start with short words (less than 8 letters). Give students the first few as a walk through. The teacher is best positioned between the Start and Accepting states. The task is to write a word made of 'a's and 'b's. If they end up finishing in the accepting state, they get a sweet. They don't if they are passing through the accepting state before the word is complete. Having finished, rejoin the end of the queue for another go. For those who catch on quickly, try words over 14 letters long.

Back in the classroom encourage children to articulate what makes acceptable words. Through pattern recognition, children generalise the rules of the machine, which they apply when choosing new words. What is the longest word in this machines language? Clearly we can't write a word with an infinite number of letters, so we need a shortcut for writing 'any number of' 'a's or 'b's. Children may be familiar with the use of wild cards, but if not, now is the time to introduce them. An asterix represents 'zero or more' of the previous character. To reinforce understanding, you could extend the FSA exercise by moving the Accepting State, adding a second Accepting State or ask students to choose words that involve the use of a wild card.



The exercise could also be repeated with the more complex machine included. This FSA introduces a 3 letter alphabet, and offers more challenge. It can be approached as a paper exercise, and is included in the photocopiable resources. Generalising from specific instances to broader rules takes practice. The presentation ends with three simple FSAs. Challenge the students to express the rules for acceptable words and ask if any can be expressed using wildcards? It's not easy. We really need tools for expressing some other rules. There are recognised shortcuts used to write general or 'regular expressions' ('regex' for short) and any regular expression can be expressed as a finite state machine. The next activity introduces regular expressions.