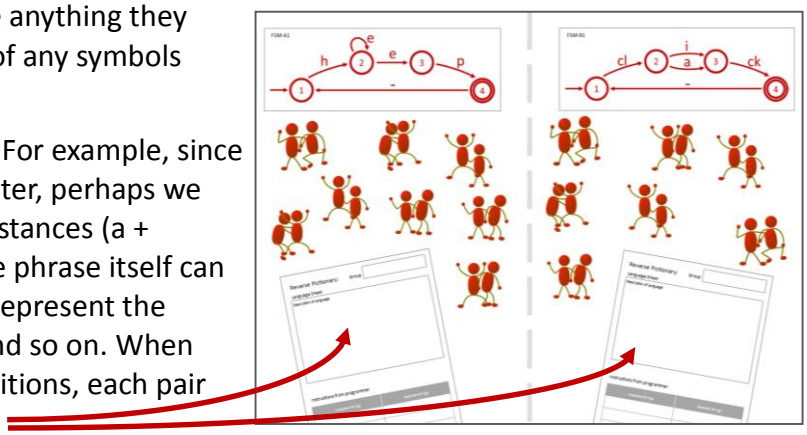
A practical activity to develop familiarity with finite-state automata (FSA) and a fun classroom demonstration. They encourage students to design their own notations for regular languages, and provide motivation for learning precise notation.

Preparation required:
Reverse Pictionary sheets for students.
A variety of props for Mr McDuff's breakfast.

Reverse Pictionary

This exercise, developed by Linda Pettigrew, comes from the CS Field Guide produced in New Zealand. Split the class in half and ask them to pair up. Give each pair in one half a copy of the FSM-A1 diagram, and those in the other half a copy FSM-B1. Give each pair a Language Sheet as well. They will be writing in the top half only. Each half should not see the FSM the other half have been given.

Give the pairs five minutes to come up with a description of the words their FSM will accept. Encourage them to think about ways to describe repeating sequences. They may be familiar with some symbols used in regular expressions, such as *, but they can use anything they decide. But they also need to list the meaning of any symbols they use so others can interpret them.



You may need to help get the students started. For example, since * represents zero or more instances of a character, perhaps we should have a symbol to mean 'one or more' instances (a + perhaps). We could then write he+p. The whole phrase itself can be repeated, with a hyphen, so how might we represent the repetition of a string rather than a character, and so on. When they are happy with their description and definitions, each pair completes the Language Description box.

Pairs then swap the language description with a pair in the other half.

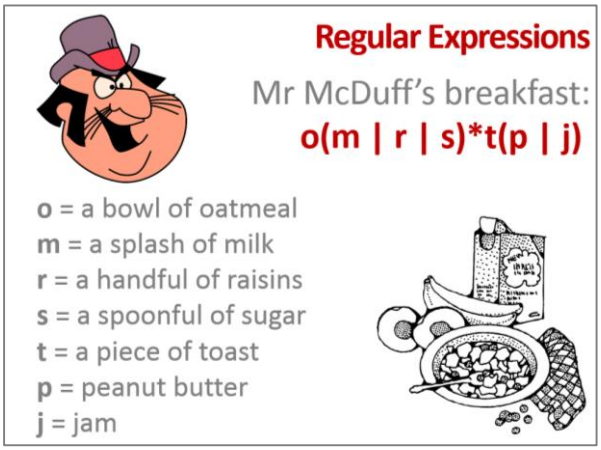
The receiving pair now complete the bottom half of the sheet. Having read the description they enter six strings they think would be accepted, and six they think would be rejected.

Finally, the sheets are gathered in and redistributed back to the original half. They do not have to return to the original pair. Each pair now acts as a 'computer' taking each input string provided at the bottom and confirming it conforms to the FSM. If a string is accepted / rejected incorrectly, the pair have to work out where the error arose.

Once familiar with the activity, a second set of FSM can be shared and the activity repeated.

Follow up discussion can investigate whether some descriptions were longer than needed, or confusing, and whether the language of the FSM was captured in the description. This provides a constructivist approach to introducing the notation of regular expressions whilst emphasising the point that any language expressed by a FSM can be represented as a regular expression.

Mister McDuff's Breakfast



An idea from the MegaMath project, Mr McDuff's Breakfast is a child friendly activity to introduce regex notation. This can be acted out to a class. Mr McDuff likes his breakfast. His choices are summarised in the items listed. Mr McDuff likes variety. Students should try to summarise the rules of his eating habits.

Make several breakfasts in front of the class until someone can explain that Mr McDuff always has a bowl of oatmeal and a piece of toast.

He sometimes has milk or raisins or sugar on his oatmeal. He may have it plain. If he has sugar, he may have several spoons, similarly with handfuls of raisins or splashes of milk. His toast will always have either jam or peanut butter. We can summarise all his breakfasts in the regular expression. Note the 'pipe' symbol (|) here signifies OR, not the distinction between input and output, as in a finite-state machine.

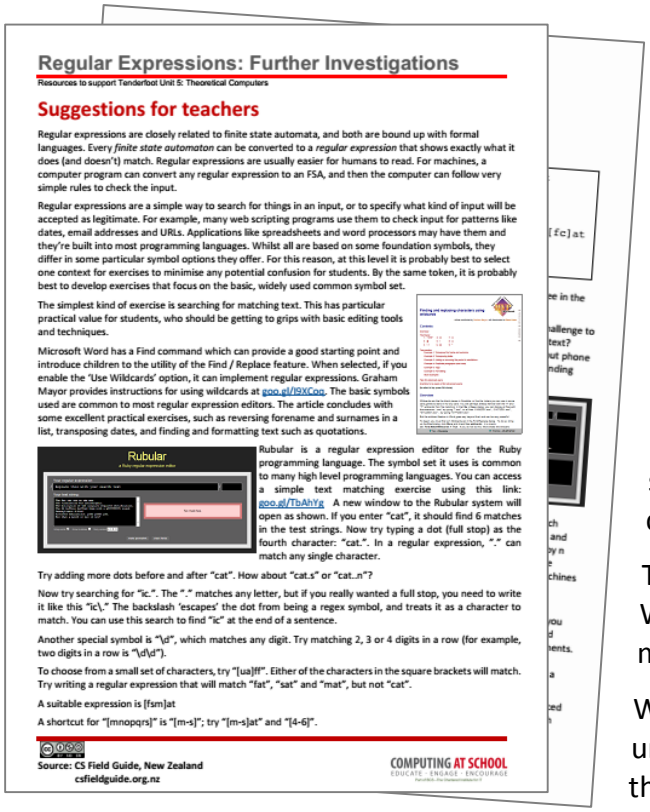
Three key symbols to emphasise:

- * represents zero or more of the preceding element
- | represents alternatives (or)
- ( ) enclose multiple items to which a symbol applies

Armed with this knowledge, can the children come up with a regular expression for their own breakfast? They should include anything they drink as well as eat.

Taking It Further

CS4Fn has a very good article aimed at secondary aged students. It introduces regular expressions by looking at knitting patterns. You can find the article via this link: goo.gl/h3lzvF. It introduces all the key notation for writing regular expressions. A good supporting homework.



Regular Expressions: Further Investigations

Resources to support Tenderfoot Unit 5: Theoretical Computers

Suggestions for teachers

Regular expressions are closely related to finite state automata, and both are bound up with formal languages. Every *finite state automaton* can be converted to a *regular expression* that shows exactly what it does (and doesn't) match. Regular expressions are usually easier for humans to read. For machines, a computer program can convert any regular expression to an FSA, and then the computer can follow very simple rules to check the input.

Regular expressions are a simple way to search for things in an input, or to specify what kind of input will be accepted as legitimate. For example, many web scripting programs use them to check input for patterns like dates, email addresses and URLs. Applications like spreadsheets and word processors may have them and they're built into most programming languages. Whilst all are based on some foundation symbols, they differ in some particular symbol options they offer. For this reason, at this level it is probably best to select one context for exercises to minimise any potential confusion for students. By the same token, it is probably best to develop exercises that focus on the basic, widely used common symbol set.

The simplest kind of exercise is searching for matching text. This has particular practical value for students, who should be getting to grips with basic editing tools and techniques.

Microsoft Word has a Find command which can provide a good starting point and introduce children to the utility of the Find / Replace feature. When selected, if you enable the 'Use Wildcards' option, it can implement regular expressions. Graham Mayor provides instructions for using wildcards at goo.gl/I9XCoq. The basic symbols used are common to most regular expression editors. The article concludes with some excellent practical exercises, such as reversing forename and surnames in a list, transposing dates, and finding and formatting text such as quotations.

Rubular

Rubular is a regular expression editor for the Ruby programming language. The symbol set it uses is common to many high level programming languages. You can access a simple text matching exercise using this link: goo.gl/TbAhYg A new window to the Rubular system will open as shown. If you enter "cat", it should find 6 matches in the test strings. Now try typing a dot (full stop) as the fourth character: "cat.". In a regular expression, "." can match any single character.

Try adding more dots before and after "cat". How about "cat.s" or "cat..n"?

Now try searching for "ic.". The "." matches any letter, but if you really wanted a full stop, you need to write it like this "ic\." The backslash 'escapes' the dot from being a regex symbol, and treats it as a character to match. You can use this search to find "ic" at the end of a sentence.

Another special symbol is "\d", which matches any digit. Try matching 2, 3 or 4 digits in a row (for example, two digits in a row is "\d\d").

To choose from a small set of characters, try "[ua]ff". Either of the characters in the square brackets will match. Try writing a regular expression that will match "fat", "sat" and "mat", but not "cat".

A suitable expression is [fsm]at

A shortcut for "[mnopqrs]" is "[m-s]"; try "[m-s]at" and "[4-6]".

Source: CS Field Guide, New Zealand
csfieldguide.org.nz

COMPUTING AT SCHOOL

RegexOne (regexone.com) is a good website for teachers to learn the basics of writing Regular Expressions. It's a little dry, and not recommended for children. However, students would benefit from some experience of writing regular expressions. In themselves they are good pattern matching exercises, an essential element in computational thinking. But practical exercises can also illustrate the sort of 'real world' tasks in which Regular Expressions (and Finite State Machines) might be used.

Regular Expressions: Further Investigations is a supplementary document which provides further suggestions and detailed exercises teachers can use in class.

These start with practical exercises using the Find feature in Word to setting your own exercises using Rubular, one of many regex checking utilities.

Whether or not these are used with students, teachers unfamiliar with this area would benefit from working through the suggestions.