Berry, M. & Csizmadia, A.P. (2016) *The silent C in STEM.* [Working paper submitted for CIDREE-STEM 2016, December 22nd].

The silent C in STEM

Miles Berry; Andrew Csizmadia

Computing at School, England

Abstract

Computer science owes its very foundation to mathematics: 2016 marks the 80th anniversary of Turing's paper on computable numbers, with an application to the Entscheidungsproblem, and historically there have been very strong connections between these two domains. In England's new (2014) national curriculum, these connections are made apparent to primary and secondary pupils through an emphasis on mathematical reasoning in the mathematics curriculum and the 'golden thread' of computational thinking which runs throughout the computing curriculum. We explore how themes such as abstraction, algorithms, decomposition, generalisation and logical reasoning can be applied to solve problems in both domains.

Much of modern mathematics draws extensively on computation, and we discuss some ways in which computer software, and in particular programming in languages such as Scratch, Snap! or Python can be used to enhance and enrich learning in mathematics. We look too at the mathematical requirements of England's computing curriculum and consider how these can be addressed through 'unplugged' and computer-based pedagogies.

We conclude with a discussion of some of the ways in which the UK computing subject association, Computing At School, has supported the implementation of the curriculum, covering initiatives such as the Network of Excellence, Barefoot Computing and Tenderfoot Computing.

Introduction

2012 was a year of transformation for computing education in English schools. At that year's BETT Show, Michael Gove, then the Secretary of State for Education, announced that Information and Communication Technology (ICT) would be disapplied from the English national curriculum[1]. He subsequently announced that it would be replaced with a more robust and rigorous academic subject, for which statutory programmes of study were produced at the end of a lengthy consultation in 2012-13[2]. In the same year, the Royal Society published a report on computing education, Shut down or restart[3]. The report made a strong case for the inclusion of computer science in the school curriculum, as a distinct academic discipline, perhaps comparable to physics.

In describing the nature of computer science, the report's authors noted:

> *Computer Science is an 'underpinning' subject, in the sense that its concepts are useful to many other science and engineering disciplines, particularly physics, and in some cases they are relied upon to such an extent that they can be considered to be part of that subject too. For example, algorithms are sometimes considered to be an element of discrete mathematics, and the logical and rigorous approach of Computer Science has much in common with mathematics in general.*

But also that

[1] Gove, M. (2012). Michael Gove speech at the BETT Show 2012 (11/1/2012)

[2] Department for Education (2013). The National Curriculum in England. London: DfE

[3] Furber, S. (2012). Shut down or restart? The way forward for computing in UK schools. The Royal Society, London.

Berry, M. & Csizmadia, A.P. (2016) *The silent C in STEM.* [Working paper submitted for CIDREE-STEM 2016, December 22nd].

> *Computer Science can sometimes suffer from being assumed to be primarily a 'tool' for other sciences rather than a subject in its own right. It is both of these, and in particular it is a science and an engineering discipline. However most STEM initiatives do not explicitly refer to Computer Science as a STEM discipline.*

Hence, the somewhat Cinderella role of computing as the 'silent C' in STEM, unlike the more pronounced 'I' of MINT.

Computational thinking and mathematical reasoning

From primary school onwards, children's arithmetic has two quite distinct stages: thinking about the question, and then working out the answer. A sum as simple as 23 + 39 demands that the child be able to decode these symbols in some meaningful way, and determine which algorithm to bring to bear in order to calculate the answer: then, and only then, can the child go on to working out the answer. When faced with a word problem, for example, 'how much change will I receive from a 5 Euro note if I buy three apples each costing 40 cents?', the same two stages apply, this time demanding a degree of abstraction as the child moves from the particular context to its mathematical representation, in this case 5 - 3 x 0.40.

More generally, we might see that most, indeed perhaps all, mathematics mirrors these two stages - thinking about problems and then manipulating symbols according to rules (i.e. a more sophisticated version of working things out). The formalist view of mathematics is that it consists of the consequences of certain string manipulation rules: for example Euclidean geometry can be thought of as those statements which can be formed by manipulating geometric axioms according to the laws of inference. Even within this formalist paradigm, practical, useful mathematics demands some thinking about which particular manipulations of strings will take us towards the solution of the problem facing us.

This view of mathematics as symbol manipulation lies at the foundation of computing. Up until the 1940s, *computer* was a job title - given to those paid to do arithmetic on paper or mechanical calculators according to the rules and procedures given them by their managers. Turing[4] expressed this symbol manipulation view of mathematics in his seminal paper, 'On computable numbers, with an application to the Entscheidungsproblem', defining computable numbers as those which could be written down by a machine and generalising this to computable functions and computable predicates: it is on this work (and the parallel work of Alonzo Church[5]) that computer science is founded, and thus the strong connections between mathematics and computer science, from primary school level up, should not surprise us.

In the decades since Turing's and Church's work, mathematics, as with many other fields, has been transformed by digital computing. The symbol-manipulation (or working out) phase of the mathematics done in science, finance, social sciences, the arts and every other domain (other than education) is done now by digital computers, rather than by people, as Wolfram explains in his TED talk[6]. Indeed, much of the symbol manipulation of even pure mathematics is now often done by digital computers rather than mathematicians, or their research assistants, themselves (see, e.g., Appel and Haken's computer-assisted proof of the Four-Colour Theorem[7].)

[4] Turing, A. M. (1936). On computable numbers, with an application to the Entscheidungsproblem. J. of Math, 58(345-363), 5.

[5] Church, A. (1936). An unsolvable problem of elementary number theory. American journal of mathematics, 58(2), 345-363.

[6] Wolfram, C. (2010). Conrad Wolfram: Teaching kids real math with computers. TED.

[7] Appel, K., & Haken, W. (1977). Every planar map is four colorable. Part I: Discharging. Illinois Journal of Mathematics, 21(3), 429-490.

The first phase of mathematics - thinking about the problem or the system - remains largely unchanged. Creative and imaginative problem solving lies at the heart of mathematics. Polya[8] suggested four phases for problem solving: understand the problem, devise a plan, carry out the plan and look back, plus a number of associated heuristics. Wolfram[9] argues convincingly that this is what mathematics education should now focus on, given that actual computation is now done by machines, and suggests his own four-stage helix for problem solving: define questions, translate to maths, computer answers and interpret. There are parallels here with the development process in software engineering: specification, design, implementation and testing. It would be wrong to think of school mathematics as being confined to manual computation. Problem solving and mathematical reasoning is an essential part of mathematics education. For example, the English National Curriculum [10] aims to ensure that all pupils:

> *"reason mathematically by following a line of enquiry, conjecturing relationships and generalisations, and developing an argument, justification or proof using mathematical language; and*
> *can solve problems by applying their mathematics to a variety of routine and non-routine problems with increasing sophistication, including breaking down problems into a series of simpler steps and persevering in seeking solutions."*

Given the strong connections between mathematics and computer science, it would be surprising if the sort of mathematical reasoning involved in understanding problems and planning their solution was not mirrored by similar thinking in specifying systems and designing solutions in the field of computing. Just as mathematics might be seen as thinking followed by symbol manipulation, so programming can be seen as algorithms plus code (cf Wirth[11]). Before programmers begin work on coding solutions, they need to have fully understood the problem and have a clear plan (an algorithm) for how to solve it.
The term 'computational thinking' has been coined to describe

> *the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent.[12]*

Whilst there is not yet a universal consensus over the exact ingredients of computational thinking[13], its importance in computing education is widely accepted. It is seen as a 'golden thread' running through the English National Curriculum for Computing[14], which begins:

> *"A high-quality computing education equips pupils to use computational thinking and creativity to understand and change the world. Computing has deep links with mathematics, science and design and technology, and provides insights into both natural and artificial systems."*

[8] Pólya, G. (1957). How to Solve It. Garden City, NY: Doubleday
[9] https://www.computerbasedmath.org/assets/img/case-for-computer-based-math-education/CBM_brochure.pdf
[10] Department for Education (2013). The National Curriculum in England. London: DfE
[11] Wirth, N. (1978). Algorithms + data structures = programs. Prentice Hall PTR.
[12] Wing, J. (2010). Computational Thinking: What and Why? http://www.cs.cmu.edu/~CompThink/resources/TheLinkWing.pdf
[13] Selby, C., & Woollard, J. (2013). Computational thinking: the developing definition.
[14] Department for Education (2013). The National Curriculum in England. London: DfE

Building on Brennan and Resnick's work[15] in which computational thinking is explored as concepts, practices and perspectives, Computing At School's 'Barefoot Computing'[16] continuing professional development programme for primary teachers identified six concepts and five approaches for computational thinking (qv Computing At School's QuickStart Computing guidance[17]). The concepts provide a unified approach to problem solving in both mathematics and computing, with a number of the example activities produced for Barefoot Computing linking these to topics in the English mathematics curriculum.

- Logical reasoning
  - In computing pupils use laws of inference to predict what programs will do from their source code, to detect and correct errors in algorithms and programs and to analyse the correctness and efficiency of algorithms; pupils learn about Boolean logic and its applications to circuits, programs and search. Program execution at the CPU level relies on logic gates.
  - Mathematics is underpinned by set theory and logic. Mathematical reasoning is fundamentally logical reasoning. In mathematics, pupils will be expected to 'show their working' and to provide a justification for their answer. They form a basic understanding of sets and their relationship, which is later formalised through Venn diagrams and the notation of set theory. They are introduced to simple proof techniques in Euclidean geometry, and will use reductio ad absurdum and induction at A level.

- Algorithms
  - From an early age, pupils learn about algorithms as sets of rules or sequences of steps for real life situations such as making a jam sandwich or tidying their classroom. They learn how other algorithms are implemented as code on digital devices. They learn that there are multiple algorithms for the same problem. They create their own algorithmic solutions to computational problems and are taught some classic algorithms for search and sort, finding greatest common divisors and testing for primality. They study greedy and divide-and-conquer algorithms in a range of contexts, including graph theory. They compare the efficiency of algorithms, in time learning to use big-O notation for this.
  - Pupils are typically taught standard algorithms for problems in arithmetic and subsequently algebra. This might be as simple as 'count out the first number of sweets; count out the second number of sweets; now count how many sweets you have' for integer addition, but will go on to include standard written algorithms for long multiplication and division, as well as methods for solving linear, simultaneous, quadratic and simple differential equations. Pupils are taught standard algorithms in other contexts, including testing for primality. They learn formulae for finding perimeters, areas and volumes, and for solving quadratic equations. Some pupils may discover their own algorithmic approaches to solving some classes of problems.
- Decomposition
  - Pupils learn to break down complex problems into smaller ones, tackling each of these in turn. Pupils learn how divide and conquer algorithms can be applied recursively, efficiently reducing the number of steps needed to solve a problem (e.g. Quicksort). Pupils make use of decomposition in their programming, using procedures, functions or objects to allow the different components of complex software to be developed and tested independently. At the hardware level, pupils come to recognise how digital

[15] Brennan, K., & Resnick, M. (2012). New frameworks for studying and assessing the development of computational thinking. In Proceedings of the 2012 annual meeting of the American Educational Research Association, Vancouver, Canada (pp. 1-25).

[16] http://barefootcas.org.uk/

[17] Berry, M. (2015). QuickStart Computing. Swindon: BCS.

devices are made of multiple, complex systems, each typically made from multiple, complex subsystems.
  - Decomposition is a powerful problem solving technique in mathematics, with pupils applying this in different contexts during their time at school. At elementary level, pupils recognise how numbers are decomposed into parts using place value, and subsequently prime factors. Simple arithmetic algorithms rely on ready familiarity with decomposition using place value. Pupils learn how the area or volume of complex shapes can be found through decomposition. Subsequently, pupils learn how vectors can be decomposed into orthogonal components and how matrices (and thus systems of linear equations) can be decomposed in a number of ways.
- Patterns and generalisation
  - In computing, pupils come to recognise standard ways to solve similar problems (for example, drawing equilateral triangles, squares and regular pentagons with a turtle), subsequently developing a general solution to a class of similar problems (in this case, drawing a regular polygon). Pupils learn to use libraries of functions developed by others rather than re-creating this code for themselves. They learn how other programmer's solutions to problems may be modified to solve similar problems. As pupils' software projects become more complex, they may make use of design patterns in their work, such as 'model-view-controller', which can be applied in a wide range of contexts from computer games to text editors.
  - Young children come to recognise patterns at an early stage in mathematics education, colouring in shapes according to a rule or deciding what will come next in a sequence. They generalise their own rules of conservation of number, shape and mass from observation. Later they are introduced to patterns in number, including the times tables as well as common sequences such as square, triangular numbers and the Fibonacci sequence. They conduct mathematical investigations, first describing the rules they discover and then expressing these in algebra, as recurrence relations and then as formulae. Pupils learn generalised algorithms or techniques - thus pupils learn the algorithm for long multiplication rather than memorising times tables to 100x100 or beyond and standard results for derivatives rather than computing each from first principles.
- Abstraction
  - For Jeanette Wing abstraction lies at the heart of computational thinking [18] and its particular form in compare science serves to distinguish computational thinking from other approaches to problem solving. Computer systems, both hardware and software, are so complex that computer scientists and software engineers established ways to *manage* this complexity, by hiding or setting to one side multiple layers of detail. Pupils might first meet the idea explicitly in the bottom of computational abstractions which model the state and behaviour of real world systems - for example the motion of a Snooker ball or the spread of an epidemic. They'll also recognise abstraction in functions, classes, libraries and APIs they use in their code: where the details of implementation are left hidden, and at times inaccessible, behind well documented specifications. They use abstraction in their mental models of computation (or 'notional machines'[19]) where the layers of user interface, compiler / interpreter, operating system and the processor's presentation layer sit between their actions and the copper and silicon of the hardware.

[18] Wing, J. M. (2008). Computational thinking and thinking about computing. Philosophical transactions of the royal society of London A: mathematical, physical and engineering sciences, 366(1881), 3717-3725.
[19] Sorva, J. (2013). Notional machines and introductory programming education. ACM Transactions on Computing Education (TOCE), 13(2), 8.

  - Abstraction is important in mathematics education too, with the curriculum taking pupils from the concrete to the abstract along a path that would be familiar still to Piaget. Even at an early age, pupils form an abstract notion of, for example, three-ness from the concrete three bears, three sweets, three books etc. They form an abstract notion of triangle or cube from the experience of particular triangles and cubes. In problem solving, pupils identify the important information in the phrasing of a question, setting to one side the less relevant or irrelevant detail. Algebra might be seen as an abstraction of number. Algebra, geometry, probability and calculus can be seen as the mathematician's approach to modelling the state and behaviour of real world systems.
- Evaluation
  - In computing, it's necessary for pupils to check whether the functions, classes and programs they write produce the results they should. It's also important that digital artefacts (including, but not limited to, programs) serve their intended purpose and are appropriate for their intended audience, and embody principles of good design. Pupils will also consider the efficiency, and indeed elegance, of their code.
  - In mathematics, pupils are taught to check their solutions, typically that numbers are broadly of the correct order of magnitude and make sense in the context of the original problem. They are also taught to check their working, that each step of their solution has been carried out correctly. Later on, they'll learn to look for logical flaws in proofs and perhaps even grasp something of the aesthetics of 'elegant' proofs.

The concepts of computational thinking can be learnt and applied in 'unplugged' activities, within and beyond computing, without the use of digital technology, as the above comparison with mathematics education illustrates, see also, e.g. CS Unplugged[20]. Wing's 'information processing agent' certainly includes digital computers, but need not be limited to such devices. That said, many would argue that 'computational thinking' can be developed particularly (perhaps most) effectively when linked explicitly to the 'computational doing' of computer programming:

> *Programming plays the same role in computer science that investigations do in maths or science. Programming animates the subject and brings computer science to life; it is creative, and engaging. It illustrates otherwise-abstract concepts in completely concrete terms. It is also an incredibly useful skill.*[21]

Papert wrote how he

> *began to see how children who had learned to program computers could use very concrete computer models to think about thinking and to learn about learning and in doing so, enhance their powers as psychologists and as epistemologists.*[22]

The experience of most primary and secondary computing teachers seems to be that computational thinking is best taught when linked directly with computer programming[23]. One could argue, as Wolfram does[24] that, if computer programs and computer programming were used more extensively in mathematics education, this would allow teachers and pupils to focus

[20] Bell, T., Witten, I. H., & Fellows, M. (2015). CS Unplugged

[21] Peyton Jones, S. (2014). Understanding the new programmes of study for computing

[22] Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc..

[23] Taub, R., Ben-Ari, M., & Armoni, M. (2009). The effect of CS unplugged on middle-school students' views of CS. ACM SIGCSE Bulletin, 41(3), 99-103.

[24] Wolfram, C. (2010). Conrad Wolfram: Teaching kids real math with computers. TED.

much more attention on developing mathematical reasoning (or perhaps 'computational thinking') rather than mere calculation skills.

Using computers in mathematics education

There are many computer programs designed to contribute to mathematics education. The majority of these are one form or other of 'drill and practice' games, based on Skinner's behaviourist theory of learning[25], cycling through stimulus (question), response (answer) and reward (score). Well designed 'games' of this form are popular with both teachers and pupils and have their place in supporting the recall of mathematical facts and allowing pupils to receive immediate feedback on whether they have or have not correctly performed mathematical operations or solved problems. Some software of this nature includes an adaptive learning feature in which a pupil's response to one or more questions will determine which future questions are asked, allowing the stream of questions to be pitched appropriately at the pupil's current level of achievement.

Software tools are now available which allow teachers (and indeed pupils themselves) to create their own computer-marked quizzes, sometimes including adaptive learning features. Creating a quiz such as this is a good introductory programming activity, for teachers' professional development and as the focus for a unit of work in computing classes. Papert observed that creating drill and practice programs had more benefit than using them:

> *It is said that the best way to learn something is to teach it. Perhaps writing a teaching program is better still in its insistence on forcing one to consider all possible misunderstandings and mistakes. I have seen children for whom doing arithmetic would have been utterly boring and alienated become passionately involved in writing programs to teach arithmetic and in the pros and cons of criticisms of one another's programs*[26]

Beyond this, there are programs which can be used, very effectively, as tools for doing mathematics, or at least for the second, working out / manipulating symbols, phase of doing mathematics. These can, and many would argue should, be fully integrated into school level mathematics education. For example:

- Dynamic geometry software such as Geogebra allows pupils to perform geometric constructions, exploring how their constructions change as initial points, lines and arcs are changed interactively to discover for themselves the invariant properties of their constructions. Modern geometry software includes the ability to measure lengths, angles and areas, to perform transformations and to link geometry and algebra through plotting points and equations on Cartesian grids. The script for a geometric construction mirrors the steps of simple algorithms. The geometric objects constructed, and their properties and classes, can provide some insight into the foundations of object oriented programming[27].
- Computer algebra systems such as Mathematica provide extensive functionality for arithmetic calculations with arbitrary precision, working with algebraic expressions, performing differentiation and integration algebraically and carrying out computation on a far broader range of objects, including data, sound, images and video. Mathematica, and the associated Wolfram Language, can be a motivating and accessible way into functional

[25] Skinner, B. F. (1938). The behavior of organisms: an experimental analysis.

[26] Papert, S. (1972). Teaching children thinking. Programmed Learning and Educational Technology, 9(5), 245-255.

[27] Alberti, M. A., Bastioli, E., & Marini, D. (1995). Towards object-oriented modelling of euclidean geometry. The Visual Computer, 11(7), 378-389.

programming. Its creator Stephen Wolfram argues that it is a tool well suited to the development of computational thinking[28].

- The humble spreadsheet, for example Microsoft's Excel, can be used to visualise and summarise data, including randomly generated data in simulations, as well as to model mathematically the state and behaviour of complex real world systems. It too provides an introduction to functional programming through its system of declarative cell-based formulae, although it is a rather limited introduction as there is no readily accessible system of higher order functions[29].
- High end statistical data analysis packages such as R, allow pupils to move beyond the visualisation and summary tools available in Excel to explore data using multi-factor analysis, statistical modelling and machine learning, and to work with 'big' data sets which can provide a highly motivating context for learning that simple class or school based data collection activities cannot.

Coding in mathematics education

Given the connections between computational thinking and mathematical reasoning discussed above, and the likelihood that the former, and thus perhaps the latter, might be best developed through some practical experience of coding, it is reasonable to explore some opportunity for using computer programming in the mathematics classroom, in a way which not only allows pupils to develop their coding skills, but also provides some insights into mathematics.
Again, Papert's work provides some foundation for this:

> *"In my vision, the child programs the computer and, in doing so, both acquires a sense of mastery over a piece of the most modern and powerful technology and establishes an intimate contact with some of the deepest ideas from science, from mathematics, and from the art of intellectual model building."*[30]

Examples of programming activities in mathematics education might include:

- Turtle graphics: an approach to geometry in which the learner programs a 'turtle' (originally a small robot, now typically on screen) to draw geometric shapes. Initially these might be simple regular polygons, with the child learning for themselves how the sum of exterior angles totals 360°, but would later include compound shapes and recursively defined fractals such as Koch Flakes and Sierpinski Triangles. Turtle graphics was introduced initially in the Logo programming language, and this remains a popular introductory implementation, although most pupils are more likely to encounter this initially in Scratch's Pen commands. Secondary school teachers have found that Python's Turtle library provides a good means to bridge the gap from block-based programming in Scratch to more general purpose text-based programming languages[31].
- There is much more to both Logo and Scratch than turtle graphics. Scratch, for example, allows scope for pupils to create 'Monte Carlo' method simulations using random numbers, running Scratch in turbo mode to perform many trials of the experiment very quickly. A classic example is the estimation of Pi by comparing the distribution of random points in a circle to those in its bounding box[32].

[28] http://blog.stephenwolfram.com/2016/09/how-to-teach-computational-thinking/

[29] Peyton Jones, S., Blackwell, A., and Burnett, M. (2003). A user-centred approach to functions in Excel. International Conference on Functional Programming (ICFP'03), Uppsala, 2003.

[30] Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. Basic Books, Inc..

[31] Dorling, M., & White, D. (2015, February). Scratch: A way to logo and python. In Proceedings of the 46th ACM Technical Symposium on Computer Science Education (pp. 191-196). ACM.

[32] https://scratch.mit.edu/projects/61893848/#editor

- Primary and secondary mathematics places great emphasis on arithmetic using fractions, which are rarely supported directly in most common programming languages (Mathematica is a notable exception here). It's instructive for pupils to develop their own functions for fractions or to create a fraction class. The overloading of operators permitted in Python (or C++) allows pupils to then use their class as they would other numerical data types. One of the first tasks in developing a system for fractions arithmetic is calculating the greatest common divisors of two numbers: pupils can investigate for themselves alternate algorithms for such a task.
- Much is gained through pupils working out solutions to problems and puzzles through unplugged, pencil and paper approaches. Much too is gained through developing programs to solve such problems and puzzles, allowing a far broader class of related problems to be explored computationally. Perhaps the real distinction between mathematical reasoning and computational thinking is that in the former we seek a solution, in the latter a process through which the solution might be found. Project Euler is a rich source of problems that might be explored from mathematical and computing perspectives.
- Simple mathematical investigations, such as exploring how many different combinations of coins can make a set amount (as discussed by Graham et al[33]), can be tackled by young pupils through a process of trial and improvement or exhaustive search. Pupils can discover for themselves the recurrence relation in such problems, and may attempt to work out tables of values by hand, or as a spreadsheet. Such recurrence relations lend themselves to implementation as recursive functions in appropriate programming languages.

Mathematics in computing

Computer science itself makes demands on pupils' mathematical knowledge. The English computing programmes of study[34] were revised substantially in the drafting process at the request of the then ministers to increase the amount of mathematical content[35]. For 11-14 year olds, the new expectations include:

- *understand how numbers can be represented in binary, and be able to carry out simple operations on binary numbers*, which is often taught through 'unplugged' approaches in which pupils take what they know of decimal place value and arithmetic algorithms in base 10 and apply these to binary numbers and arithmetic in base 2. The standard algorithm for multiplication in base 2 is very similar to the 'Russian peasant' method for multiplication by doubling (ie a binary shift left) and selective addition, and binary long division offers a significant reduction in cognitive load compared to the equivalent base 10 version.
- *understand how data of various types (including text, sounds and pictures) can be represented and manipulated digitally, in the form of binary digits*: manipulation here might include shifting upper case to lower case by subtracting 32 from US-ASCII values, increasing or decreasing the volume of music by multiplying or dividing individual audio sample values, or brightening, darkening or reducing the colour depth of images through applying formulae to the bitmap. Pencil and paper exercises can be used here, but so can spreadsheets or text-based programming languages.
- *understand simple Boolean logic [for example, AND, OR and NOT] and some of its uses in circuits and programming:* pupils are introduced to truth tables; they also explore simple logic circuits, sometimes using online simulators, but often through pencil and paper exercises, recognising how binary addition can be accomplished through such circuits; the link between set theory and Boolean operators in search results is explored.

[33] Graham, R. L., Knuth, D. E., & Patashnik, O. (1989). Concrete Mathematics. Massachusetts: Addison-Wesley.

[34] Department for Education (2013). The National Curriculum in England. London: DfE

[35] Bannister, P. and Berry, M. (2013). Curriculum reform and computing. Presentation at Westminster Education Forum, 28/2/2013

- *understand several key algorithms that reflect computational thinking; use logical reasoning to compare the utility of alternative algorithms for the same problem*: pupils learn about algorithms for search and sort and, often, for some aspects of number theory, such as testing for primality or finding the greatest common divisor; they compare algorithms for time efficiency, although Big-O notation is typically left until elective qualifications at 16-18. A nice way to illustrate search algorithms is through playing 'guess my number' games; search algorithms are often taught using 'unplugged' approaches such as ordering weights using comparisons with a pan balance.
- *make appropriate use of data structures*: pupils learn about variables between the ages of 7-11. At age 11-14, they study lists and arrays: unplugged activities with lists might include different approaches to shuffling decks of cards, itself a useful topic for mathematical investigation. Some teachers choose to introduce elementary graph theory at this point, for example demonstrating how seemingly different problems may be represented using the same graph or setting pupils the challenge of finding a minimum spanning tree for a graph using a greedy algorithm such as Prim's[36].
- *understand a range of ways to use technology […] securely, including protecting their online identity and privacy*: some teacher use this requirement as an opportunity to teach pupils about cryptography, including public-private key systems, illustrating this with the relative difficulty of factorising large numbers compared to the ease with which prime numbers of a similar size can be multiplied.

Computing At School

Since its beginning in 2008, Computing At School (CAS, the UK subject association for computer science) has striven to support computing teachers in developing their subject knowledge, pedagogic practice and confidence in computing. CAS initiatives of particular relevance here are the Network of Teaching Excellence in Computer Science, Barefoot Computing, Tenderfoot Computing, QuickStart Computing, Project Quantum and CAS's Computational Thinking Working Group.

- The Network of Teaching Excellence in Computer Science (NoE), established in 2012, funded by the English Department for Education and administered by CAS, is a national community of professional practice. It draws its membership from teachers in schools and university academics, and focuses its support on the teaching of computing in state-funded English schools. In the period up to December 2016, the NoE provided over 56,000 instances of continuous professional development (CPD)[37]. These have been provided by CAS Master Teachers, university partners and CAS Hubs through formal training events including regional and national conferences, mentoring, coaching, peer observation, and peer collaboration in the development of resources and assessment. Since November 2015, ten university based NoE regional centres have ensured national coverage and consolidate and expand the network.
- The Barefoot Computing Project[38] began in 2013. It was initially funded by the Department of Education and subsequently by BT. Barefoot Computing took the view that primary computing teachers only needed to know enough computer science to teach the primary computing curriculum. It assumed that they knew already how to teach and how to use technology, but simply lacked computer science subject knowledge. It developed a set of concept guides to computer science topics, including a detailed treatment of computational

[36] Prim, R. C. (1957). Shortest connection networks and some generalizations. Bell system technical journal, 36(6), 1389-1401.

[37] BCS/CAS (2015). Network of Teaching Excellence in Computer Science DfE Project Quarterly Review December 2015. Swindon: BCS

[38] http://barefootcas.org.uk/

thinking as discussed above. These were supplemented by exemplar lesson plans in which primary teachers are shown how these computer science concepts could be linked to other topics in the English primary curriculum. For example, in the Bee-Bots 1,2,3 programming activity[39], pupils create an algorithm to draw the shape of a numeral, then they program a Bee-bot floor robot to navigate a route tracing the shape. The resources are freely available online, but are introduced to primary teachers via free workshops led by trained volunteers, including BT's own staff.

- QuickStart computing[40], funded jointly by Microsoft and the Department for Education, sought to develop quality continuing professional development materials for CAS Master Teachers and Hub leaders to use in meetings they organised, as well as providing resources on which primary computing coordinators and secondary heads of computing could draw for providing CPD for teachers in their schools. The primary pack focussed on computing subject knowledge, whilst the secondary pack focussed on issues of planning, teaching and assessing computing. Both packs included pointers to further reading, activities and resources.
- The CAS Tenderfoot[41] project began in 2015, funded by Google. It is a CPD programme for secondary computing teachers. It covers the background computer science subject knowledge, other than programming, that secondary computing teachers might need to teach the 11-14 computing curriculum, and, like Barefoot Computing, has a focus on the development and application of computational thinking. Topics available to-date include: bits and bytes, finite state machines and simulation. One approach used by Tenderfoot is that of puzzle-based learning in which puzzles are presented for learners to solve, learners' solutions are discussed and then the teacher presents a solution which leads to a discussion of the computer science or mathematical concept that underpins the solution. For example, the puzzle of the movement of a knight around a chessboard visiting each square only once can be solved by the use of graph theory and in particular the identification of a Hamilitonian cycle [42]. This project will provide high quality teaching resources and activities that teachers can adapt or adopt for their own pupils.
- Project Quantum[43]. Funded by ARM, Microsoft and Google, Quantum seeks to crowd-source a national item bank of multiple choice questions for assessing computing, including computational thinking. Some 2,100 questions are currently available, and as increasing numbers of pupils undertake questions, statistical (Rasch[44]) analysis of individual learners performance on each item will allow high quality questions and common misconceptions to be reliably identified.
- CAS convenes a number of working groups. These groups consist of subject matter experts drawn from computing teachers, educationalists and computer scientists. One of these groups, the Computational Thinking Working Group, sought to articulate, amplify and advocate what computational thinking is and provide advice and guidance for both preservice and inservice computing teachers on computational thinking and how it might

[39] http://barefootcas.org.uk/barefoot-primary-computing-resources/concepts/programming/ks1-bee-bots-12-3-programming-activity/ (free registration required)

[40] http://www.quickstartcomputing.org/

[41] https://www.computingatschool.org.uk/tenderfoot/

[42] Curzon, P. (2014). A Brief Tour of Computational Thinking: The Knight's Tour and Other Puzzles. London: Queen Mary Univesity of London.

[43] Oates, T., Coe, R., Peyton Jones, S., Scratcherd, T., and Woodhead, S. (2016). Quantum: tests worth teaching to. Cambridge: Computing At School.

[44] Rasch, G. (1993). Probabilistic models for some intelligence and attainment tests. MESA Press.

best be taught in schools. The group has produced a guide to computational thinking for teachers[45].

Conclusion

There is a symbiotic relationship between mathematics and computer science, at least to the extent that mathematics underpins computer science and computing can be used to automate much of mathematics. Beyond this, there appear to be strong parallels between computational thinking and mathematical reasoning: problem solving in either domain might usefully draw on a common vocabulary of concepts, such as that developed for some of CAS's projects.

[45] Csizmadia, A., Curzon, P., Dorling, M., Humphreys, S., Ng, T., Selby, C., & Woollard, J. (2015). Computational thinking: A guide for teachers.