Draft: Under Development - 20 Sep 2012

What is computational thinking?

Who needs it?

Why?

How can it be learnt?

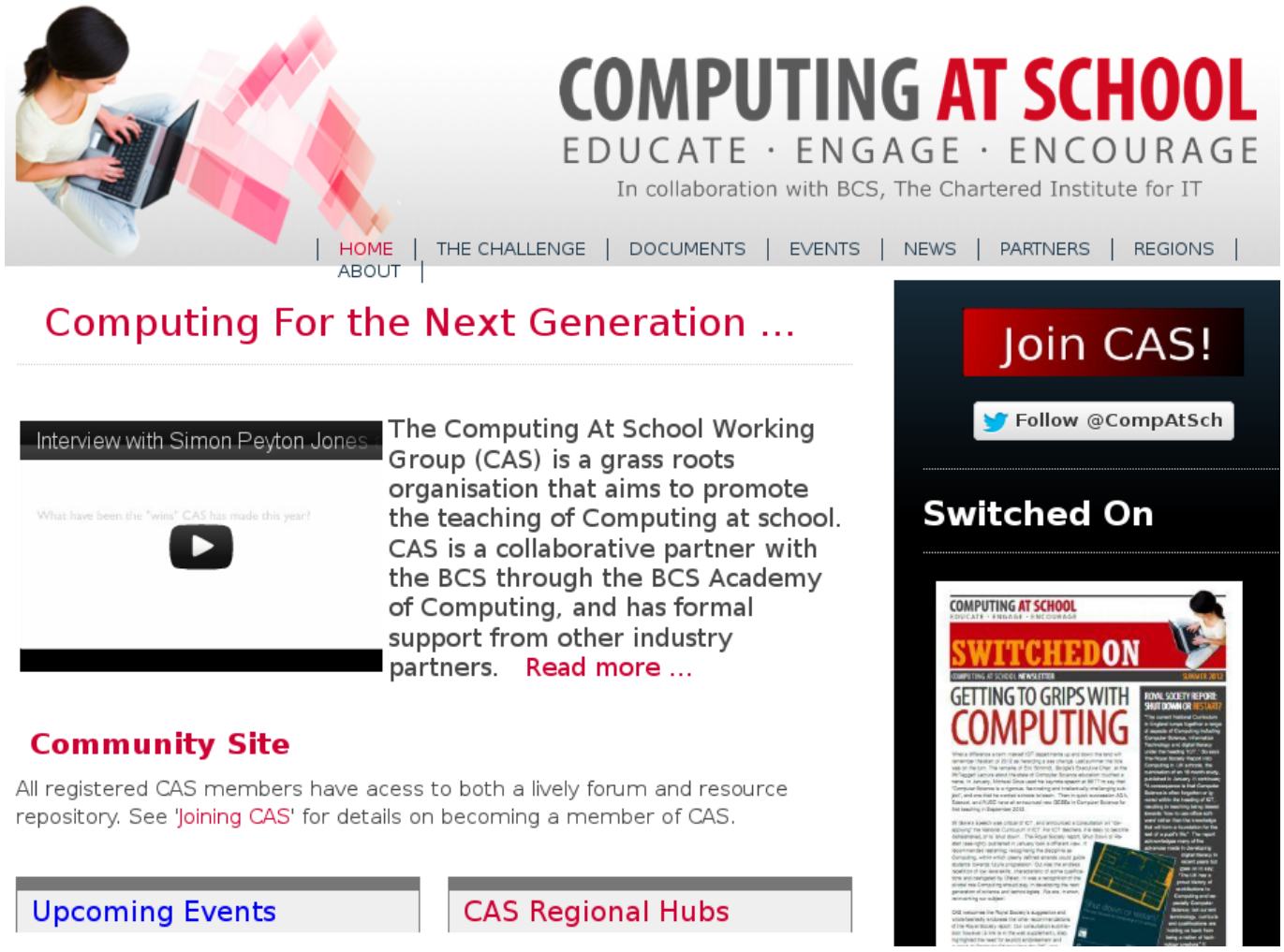
(Can it be taught?)

Aaron Sloman
http://www.cs.bham.ac.uk/~axs/

These slides will be added to my 'talks' directory:
http://tinyurl.com/BhamCog/talks/#alt2012

and slideshare.net: http://slideshare.net/asloman

Any members of CAS here?



I found it interesting, and a little surprising, that very few people at the learning technology meeting were also in the Computing at School group.

See `http://www.computingatschool.org.uk/`

Original Abstract

This was the original abstract posted on the ALT conference web site.

Computational thinking goes beyond programming and is hard to teach well, but can help us understand natural and artificial information-processing systems, including human minds.

I shall explain why an extended version of Jeannette Wing's notion of 'computational thinking' is a requirement, not just for IT professionals, but also for scientists, philosophers, and others trying to understand our world, including human minds and other products of biological, and cultural evolution.

Computational thinking goes far beyond programming and is not easy to teach.

Current discussions about computing education mostly aim to produce high calibre application developers, ignoring the need to educate outstanding scientists and thinkers, including philosophers, who need to learn new, computationally informed, ways of looking at old things, such as behaviours of microbes, insects, toddlers and economies.

Developing technology to support that learning will not be easy, but some first steps will be illustrated.

Revised Abstract – for non Computer Scientists

- **What is computational thinking?**
  Seeing the universe as made of matter, energy and information, all interacting; trying to understand how they interact, building and testing explanatory theories, and using that understanding in many fields of study:

  understanding evolution, understanding learning, understanding emotions, understanding education, understanding the economy, understanding ecosystems, ... all in terms of their information processing mechanisms and functionality.

- **Who needs it?**
  Biologists, psychologists, neuroscientists, psychotherapists, social scientists, philosophers, educators, designers of educational technology, art historians, theologians, politicians, managers, ... parents with young children, ...

- **Why?**
  Because without computational thinking you will use shallow and inadequate models, do shallow and inadequate empirical research (like alchemists before chemistry was built on a theory of the sub-atomic structure of matter (Sloman, 2012)), develop misguided educational strategies, and fail to prepare our young learners for the rest of the 21st century.

- **How can it be learnt?**
  There's no easy way: learn to build ever more complex models of natural information processing systems, try to understand the engineering design problems that evolution addressed; always ask questions about what information is needed, why it is needed, what information is available, how is it acquired, how is it interpreted, analysed, stored, used, accessed, combined with other information, used to derive new information, ... in what ways are those processes and mechanisms faulty or inadequate?
  See `http://tinyurl.com/BhamCog/misc/design-based-approach.html`

Note on these slides

ALT is the Association for Learning Technology (http://www.alt.ac.uk/about-alt) Its annual conference ALT-C 2012: "A confrontation with reality" was held at the University of Manchester, UK, 11 - 13 September 2012 (http://www.alt.ac.uk/altc2012)

Seb Schmoller, whom I met via the ComputingAtSchool email list, and who was previously the chief executive of ALT, visited me in Birmingham in March 2012. We got on well, and he invited me to give a talk at ALT-C. I accepted, with substantial doubts as to my ability to engage with ALT members, because of my ignorance of the field. When I later mentioned I had written a note on education research as alchemy he arranged for it to be published in the June issue of the ALT Newsletter (Sloman, 2012).

My talk was presented in a 30 minute session on Tuesday 11th September, and, despite having been up most of the night preparing slides, I talked without them because of a strange projector compatibility problem – despite a successful test earlier that morning. I suspect the presentation without slides was much better than anything I could have done with slides. (A video recording will be available at the ALT conference web site.)

As promised, after the conference I tried to re-organise them, partly in the hope of showing the relevance of computational thinking to ALT practitioners and researchers.

Without a deep understanding of the information processing architectures, mechanisms, and capabilities, of learners at various stages of development, educationalists will probably fail to fulfil the great educational potential of new and old technology. And they won't notice.

Jeannette Wing's paper

For people who don't know Jeanette Wing's inspirational paper "Computational Thinking" (Wing, 2006), a free version is available here: `http://www.cs.cmu.edu/~wing/`

Stuart Wray commented recently:

> "Her notion of 'computational thinking' describes a world view looking out from core computer science, with the aim of demonstrating to the world that it really can do useful things for people other than just 'geeks'". `http://www.stuartwray.net/philosophy-of-knowledge.pdf`

Yes: despite the great ideas in the paper, a CS-based world view risks missing the depth and variety of computational processes (information processing) in our universe, some presented here:
`http://tinyurl.com/BhamCog/misc/meta-morphogenesis.html`

Learners and teachers need to understand that the universe contains matter, energy, and information, and things made out of them.

Without that understanding, many phenomena can appear mysterious and inexplicable, and will be thought about in inappropriate ways, including life, mind, and learning: whose mechanisms can be misrepresented for religious, political, ideological, or other reasons, or just out of ignorance.

Ideally, all educators need to learn computational thinking also – to understand the processes going on in their pupils and what they are achieving, well or badly.

I offered related ideas in Chapter 1 of (Sloman, 1978) and in a UKCRC conference paper in 2004 `http://tinyurl.com/BhamCog/misc/gc-ed-sloman/` (A New Kind of Liberal Education)

Decades old work by John Holt, Seymour Papert, Marvin Minsky and Alan Kay is also very relevant – e.g. some aspects of the LOGO project, and aspects of the Montessori approach to education(?).

This presentation gives only a brief and partial introduction to these ideas.

Broadening Jeannette Wing's vision

Jeannette Wing:

"Computational thinking confronts the riddle of machine intelligence: What can humans do better than computers? and What can computers do better than humans?"

Compare this kind of computational thinking:

"Computational thinking addresses the nature of all intelligence, including microbe intelligence, animal intelligence, human intelligence, and machine intelligence."

What sorts of information processing do the many varieties of organisms need?

Including microbes, plants, mice, monkeys, humans, squirrels, crows, ...

What sorts of information processing can brains, and other biological mechanisms support: what do they do and how do they do it?

How can we build and test working theories of how they do it?

Can all biological forms of information processing be modelled on computers?

What alternatives are there?

How do natural information processing competences and mechanisms change/develop?

Is there something special about chemistry-based computation?

We could add

What sorts of information processing do **socio-economic** systems do?

What sorts of information processing do **eco-systems** do?

CS and AI have deep implications for Biology

Researchers have found overlaps between AI and work in developmental psychology and animal cognition.

But there are two directions of influence:

Biologically-Inspired AI (BI-AI)
vs the less noticed alternative:
AI-Inspired Biology (AI-IB)

– the former is mostly much shallower)
(There have been several workshops and conferences on AIIB in the last few years – not all using that label.)

Explaining squirrel intelligence is a deep challenge
Grey squirrels defeat many "squirrel-proof" bird-feeders – e.g. the squirrel raiding our bird-feeder held by suckers high on a patio door.

It managed to climb up the plastic-covered door frame just visible on left then launch itself sideways across the glass, landing on the tray with nuts – a remarkable piece of creative intelligence – and ballistic launch control.

They seem to be able to reason about what to do in advance of doing it, even in novel situations, requiring a primitive form of "theorem proving" about what can work?
(See Craik's ideas on next slide.)

Robotic understanding of affordances is currently far inferior to animal understanding. See this presentation extending Gibson's ideas about affordances `http://tinyurl.com/BhamCog/talks/#gibson`

Craik on Animal Computational Thinking

Kenneth Craik (Craik, 1943) suggested that in order not to have to discover empirically that certain actions would be harmful, or fatal, some animal species had evolved the ability to build models of novel situations and "run" them to work out consequences of possible actions.

I don't know whether he understood the importance of perception of and reasoning about affordances as described by Gibson (Gibson, 1979) and generalised in these slides
`http://tinyurl.com/BhamCog/talks/#gibson`.

Or whether he understood the requirement to reason not only about possibilities specified in total detail (how to jump from here to that branch) but also to use abstractions that covered classes of cases – e.g. anticipating a set of possible futures, unlike simulations in a game engine which generate exactly one future on each run, from each initial state.

The squirrel shown in the previous slide could not have simulated in advance all its sensory and motor signals and their effects in order to select the climbing strategy: it would have to cope with an explosive **continuum** (or collection of **branching continua**) of possibilities.

The need for abstraction and chunking (discretisation) of affordances and possible consequences of actions, allowing **finite** case analysis to suffice to cover **infinite** collections of future possibilities (e.g. continuously variable types of movement), may have been an important precursor of mathematical reasoning (e.g. in Euclidean geometry):
`http://tinyurl.com/BhamCog/misc/triangle-theorem.html`

Computational Thinking in Philosophy (DRAFT)

Soon after Max Clowes (Sloman, 1984b) introduced me to programming and AI around 1969-70 I began to realise that research in AI (a form of computational thinking) could lead to deep new insights in philosophy.

Examples:

Philosophy of mind and consciousness

`http://tinyurl.com/BhamCog/talks/#talk84`

Using virtual machinery to bridge the "explanatory gap"

Or: Helping Darwin: How to Think About Evolution of Consciousness

Or: How could evolution (or anything else) get ghosts into machines?

Philosophy and psychology of mathematics

`http://tinyurl.com/BhamCog/talks/#toddler`

`http://tinyurl.com/BhamCog/misc/triangle-theorem.html`

General

`http://tinyurl.com/BhamCog/crp`

The Computer Revolution in Philosophy: Philosophy, science and models of mind.

(1978 book online now)

Also on kindle for $1.98 `http://www.amazon.com/Aaron-Sloman/e/B001HQ4Q88`

Margaret Boden on Creativity

Creative Mind: Myths and Mechanisms: 2nd Edition 2004

`http://www.amazon.com/The-Creative-Mind-Margaret-Boden/product-reviews/0747411506/ref=dp_top_cm_cr_acr_txt?ie=UTF8&showViewpoints=1`

More details on computational thinking in philosophy to be added later

Types of programming worth teaching in schools

Learning to program can give learners experience of designing, testing, de-bugging, analysing, comparing and contrasting working systems – i.e. computational thinking.

This can provide a deep educational supplement to learning reading, writing, arithmetic, art, music, sewing, knitting, gardening, chemistry, physics, playing with construction kits (lego, tinker-toys, meccano, fischer-teknik, etc.), or learning woodwork

`http://www.woodworkshop.co.uk/about_the_courses.htm`

A draft list of types of programming worth teaching, to meet different educational and vocational needs is here `http://tinyurl.com/thinky-ex/kinds` – partly intended as an antidote to the very narrowly focused aims and objectives encountered in recent discussions of computing at school.

A modified version of that list is included in later slides, below. The online list is likely to be updated.

A deeply important kind of programming that is under-represented in computing education (and ComputingAtSchool discussions) is 'thinky' programming, illustrated here:

`http://tinyurl.com/thinky-ex`.

Thinky programming provides opportunities to learn new ways of thinking about how minds work (including both natural and artificial minds), both when acting in isolation and when part of a social system. E.g. thinky programs include models of holding a conversation, cooperating in building things, using pheromone trails to explore terrain, playing a game, discovering a theorem, learning to think.

An example: Debates about teaching reading

There are recurring debates about how to teach children to read, by debaters who have never designed or debugged a machine that is capable of visual perception, language production, language understanding, thinking, learning, or reading stories.

In particular, uninformed battles rage on whether to use

- phonics (associating letters with particular sounds, along with rules for exceptions), or
- "look-and-say" methods (teaching children to recognize whole words)

often based on politics, guesswork, partial evidence, personal experience, wishful thinking.

The arguments are often well-meant (though there are also vested interests), but they address an important issue that cannot be resolved without deep new theories about the information-processing functions involved in reading, thinking, and speaking, and the information-processing mechanisms that make those processes possible.

A 1976 discussion of this issue by Lee Goeller is worth reading:
`http://www.leegoeller.com/IvsR/IvR-1.htm`

Experiments aimed at evaluating alternatives are often more like alchemy than chemistry, as explained in (Sloman, 2012).

Computational analysis of reading processes suggests that both methods are required for a machine (biological or non-biological) to learn to read and understand written words.
Different languages require partly different mechanisms. Think of signing languages, for example.

Broadening the information-processing context

How often do proponents of one or other mode of teaching beginners reading consider the later stages of reading expertise?

- An expert reader does not produce noises while reading: words and phrases are linked internally to meanings. Think computationally: How can that happen?
- If understanding a linguistic medium always required translation to another medium there would be infinitely many intermediate linguistic stages and no understanding.
- When understanding speech, we don't have to learn to associate the words with other kinds of perceivable symbols (on paper, voiced, signed manually, or anything else): some comprehension must use purely internal processes – but what sort???
- A child learning to understand a spoken language learns to associate the sounds with meanings – of many kinds.

  (Compare: adjectives, nouns, pronouns, adverbs, prepositions, verbs.)
- A child learning to read is primarily learning to associate the written words with meanings: forcing that process to go **only** through another medium is based on a seriously flawed theory of what understanding is, even if it sometimes helps.
- For many children who have started reading, the processes of extending the written and spoken vocabularies grow **in parallel**, sometimes with strange results.

  (E.g. I learnt to understood "rogue" while thinking it was pronounced to rhyme with "log yew".)
- While we lack **theories** of language learning that are testable in working models, policies based on **over-interpreted empirical results** risk seriously harming children, by forcing teachers and parents to focus on mechanisms instead of inspiring learning.

That was just a tiny taste of computational thinking

But a very shallow taste.

There's a lot more to be said about

- what needs to be explained,
- what sorts of explanatory models might provide an explanation,
- whether the explanation could be tested,
  and if so
- how it could be tested,
- and if not ...

But this is not the place, and far more time is required than one presentation.

Conjecture:

Kids now learning to program in primary school, if inspired well, will grow up to understand the issues currently being debated better than their teachers, their parents, their educational administrators, or their politicians.

Can computational thinking be taught?

Yes but there are several different varieties, and we should not teach all our learners the same small subset: that will starve the culture of important ideas not taught.

In particular, don't always choose programming languages, development tools, and programming examples as if you were educating future software engineers, applicants for computer science at university, or "app" developers having fun.

Allow gifted teachers to use powerful multi-paradigm languages to construct new sub-languages (or micro-languages) relevant to different problem domains

This was the teaching philosophy behind much of the development of Pop11 and Poplog at Sussex then Birmingham university — between about 1976 and 2000
See "Beginners need powerful systems" (Sloman, 1984a)
`http://tinyurl.com/BhamCog/81-95.html#45`,
and ideas demonstrated in these online video tutorials: `http://tinyurl.com/PopVidTut`

For example: some learners, who might be future educators, should be able to create and play with grammars, parsers, and other mechanisms relevant to production and analysis of sentences, or simple stories, or simple poems.

Other examples of "thinky" programming are here: `http://tinyurl.com/thinky-ex`

including chat-bots of varying sophistication, toy expert systems for solving problems, image analysis and interpretation mechanisms, mechanisms for reasoning by analogy, simple mathematical reasoners, simple 'minds' for robots (physical or simulated) in environments of varying complexity, various sorts of game playing, planning, teaching, or modelling tools. (More on this below.)

Draft list of kinds of programming worth teaching

- **Bumpy Programming:**
  Many tasks designed for very young learners fit into this category. These are concerned with controlling movements and changing appearances of real or simulated mobile devices, like the LOGO physical or graphical turtle, or interactive video-based game programs involving bumping, shooting, eating, avoiding, etc., often with accompanying swishes, bangs, thumps and other entertaining noises. The tasks and the programming environments supporting these can vary from very elementary to professional.

  The phrase “bumpy program” is a little too specific for this general category, since I don’t want to imply that all the instances involve objects moving around and making contact with other objects. Alternatives might be “fun program”, “gee-whizzy program”, “entertaining program”. Suggestions welcome. It should probably be sub-divided.

- **Numbery Programming:**
  The oldest type of computer program – adding, multiplying, averaging and performing statistical operations on one or more numbers, doing matrix and vector operations, equation solving operations, as well as drawing graphs of numerical functions. All very useful, and getting more diverse.

- **Gadgety Programming**
  These are programming tasks concerned with controlling devices outside a computer (e.g. the Arduino). Some gadgets merely respond to output from the computer, e.g. a gadget that flashes lights or makes noises under the control of the computer. Others send signals to the computer from sensors in the gadget or buttons, or knobs, or levers on the gadget that a user can manipulate. Some gadgets may be used to control other machines, e.g. a digital thermostat that controls a heater and is itself controlled by a computer.

  A website illustrating this category is: `http://www.netmf.com/gadgeteer/`

- **Arty Programming**

  E.g. for generating poetry, stories, pictures, music, dancing robots and other works of art?

  There are several kinds of programs designed to produce something funny, e.g. puns, acronyms, limericks, etc. These could be put in a separate category or treated as a sub-category of “arty” programs.

  Arty programs can overlap with some sorts of ‘Thinky’ programs, below, depending on how much of the design, or creation, is done by the program, as opposed to the user of the program (as in creating a PowerPoint presentation). Richard Dawkins and others have shown how evolutionary computation, driven by user preferences, can create interesting “arty” results, e.g. pictures or music.
  (See work by Harold Cohen, David Cope, Simon Coulton, and others.) An introduction to generating Haikus is `http://tinyurl.com/PopLog/teach/haiku-examples.txt`

- **Presentation programs**

  There are many tools that enable a human designer or author to create something to be presented to others: a poster, a musical performance, a presentation, a document to be read, an advertisement, etc.

  Some of these merely take in very explicit instructions and obey those instructions, for example a WYSIWYG text formatter where the user is constantly assembling the text, specifying spacing, indentation, font style and size, where the line breaks should be, as well as the actual content. Others allow the user to specify, in a “markup” language, what needs to be done at some level of abstraction, leaving it to the program to work out how to achieve that, e.g. where to break lines to produce left and right aligned text, or which numbers to assign to numbered paragraphs or figures. Simple examples include html, nroff, and others that were widely used in past decades, while more complex examples, some of which are extendable by sophisticated user-defined macros, include Troff, LaTeX, PHP, JavaScript and other systems used in web design.

  Some presentation tools are internally very sophisticated, but give the user only very shallow modes of composition, too shallow for the purpose of teaching programming, or computational thinking, while others can present programming challenges of high intellectual difficulty. (E.g. advanced uses of LaTeX.)

- **Lifey Programming**
  These are programs written using 2-D patterns in a 2-D grid, which generate new patterns. The most famous example is Conway's 'Game of Life'. There is an excellent overview, along with a superb interactive Java applet here `http://www.bitstorm.org/gameoflife/` Another is: `http://www.ibiblio.org/lifepatterns` (use "Enjoy Life" button on top left).
  Draft notes on the 'Life programming language': `http://tinyurl.com/thinky-ex/life.html`
- **Modelling Programming**
  These are attempts to model the structure or behaviour of some previously existing system – e.g. the solar system, ant foraging, insect swarms, water pouring out of a jug, sand castles collapsing, traffic "pressure waves" on a motorway, traffic flow in a busy city, financial transactions, and many more. Some of the tasks for learners can be fairly elementary, especially if based on well-designed libraries. Others will tax even advanced researchers, e.g. modelling weather patterns.

  Many modern sciences make heavy use of programs, often constructed for the purpose by scientists, that model some naturally occurring phenomenon, or an aspect of human social or economic activity. So learning to build a variety of types of modelling programs can be useful for many careers in science, engineering and management.

  Some modelling programs are concerned with 'thinky' systems, described below. Some are part of a gadgety program: e.g. a program that controls an external machine or device, and chooses control strategies partly on the basis of modelling the behaviour of the device, to predict consequences of alternative strategies.
- **Gamey Programming:**
  These are programs that either play a game with a user (e.g. chess, Go-Moku, Nim) or support a game being played by two or more players (e.g. managing a game of chess, or monopoly) or present users with some puzzle solving activity, or test of some kind of skill, e.g. Solitaire, or Tetris. This category can overlap with others, especially "thinky" programming where the computer has to compete with a user by making plans, choosing moves, solving problems, or even learning to play better.

- **Exploratory Programming** (Many kinds)

  This notion is orthogonal to the other types of programming: any of them could include exploratory programming, but need not. Implementing a well understood solution is not exploratory programming.

  Programming is exploratory when programs are not written with some well defined initial specification of what is required, against which the programs are to be evaluated.

  Instead, exploratory programming is used as a way of trying out ideas, to see what happens.

  In deeper versions, this may be a way of gaining an understanding of a class of structures or processes, for example, studying ways in which polygons can vary under various sorts of constraints, or ways in which algebraic expressions relate to the shapes of graphs, or investigating the proportion of algebraic structures of a certain type that also have some specific property.

  This is often a good way to explore a class of mathematical structures (e.g. logical functions, or random walks), a class of algorithms (e.g. image interpretation algorithms) or possible ways of developing or extending some sort of tool. For example, if you write a program that generates lines in 2-D display, and involves various numbers corresponding to lengths, angles, numbers of times something is done, you can experiment with ways of changing the numbers in order to understand the set of possibilities.

  In order to explore ideas about numbers you can try pretending that the computer can manipulate lists of symbols, e.g. [a], [a a], [a a a], ... but has no arithmetical capabilities, and then try to program it to understand numbers and arithmetical operations. This can help a learner acquire a new deep understanding of the nature of numbers. (See `http://tinyurl.com/PopLog/teach/teachnums`)

  In order to investigate ideas about probabilities you can use a computer to systematically generate a collection of cases (e.g. the sets of numbers you can get by throwing three dice) and then work out the probabilities of various combinations (e.g. combinations of three numbers that add up to five) by seeing what proportion of the whole set of possible combinations includes combinations totalling five.

  In that case the computer can generate the sets, count their elements and find the ratios. It can also plot graphs displaying, for example, how the probability of getting a throw adding up to N varies with the number of dice thrown.

Someone playing with such a program can notice something that was not planned and then change the program to explore that new feature.

As with many toys, and many kinds of mathematical or scientific research, what you experience while "playing" or "exploring" can suggest new questions, new things to try, new hypotheses to test, new explanations of previously discovered patterns, or refutations of conjectures (e.g. by producing an unexpected counter example).

I suspect these are "architecture-based" rather than "reward-based" motivations. (Sloman, 2009)

An example of exploration of random walks using Pop11 is here:
`http://tinyurl.com/PopVidTut#pop11-graphics-demo.ogv`

An example of exploratory programming related to Russell's Paradox and the paradox of the barber who shaves all and only those shavers who do not shave themselves is here:
`http://tinyurl.com/thinky-ex/barber.html`

Unfortunately educational practices that take students through a collection of set tasks may suppress the development of deep exploratory skills required for discovering new, interesting, tasks.

Alas, some examining bodies require A-level computing projects to have a "user" – focusing attention on computing as engineering, whereas for many projects, including exploratory projects and scientific modelling projects, requirements come not from the (possibly arbitrary) requirements of somebody else, but from the learner's analysis of a problem, and assessment can be related to whether and how the problem was solved, not whether a user is satisfied.

Turing's exploration of biological morphogenesis (Turing, 1952) had no "user".

He was trying to test out a hunch about how microscopic patterns of interaction between two chemicals could produce visible patterns on animals and plants.

A supervisor, in the traditional sense will be needed, rather than a user.

Calling a supervisor a "user" helps to reinforce out of date and inappropriate attitudes to computing.

- **Utility ("Application" or "Appy") Programming:**
  A great deal of human effort now goes into making utility programs. These may be simple but frequently used applications e.g. a program to set an alarm to go off after a set time, or very complex utilities written to solve a particular problem, e.g. searching for a way of decoding a particular coded message.

  The wide-spread use of programmable smart phones, supported by mechanisms for making new "apps" available to many users (possibly at a cost) has encouraged many teachers and learners to focus on "appy" programming. This category overlaps with most of the others insofar as the utilities developed may serve any of the purposes listed here, including teaching programming.

  On a larger scale, utility programming, or application programming goes far beyond what can be accommodated in primary and secondary education, including design and implementation of compilers, interpreters, device drivers, operating systems (and their many components), networking systems, security systems, financial services, remote servers of many kinds, and a wide range of systems for controlling machinery including chemical plants, airliners, power distribution systems, and many more. In some cases it may be possible for teachers to design projects around "toy" versions of such systems.
- **Teachy (tutorial) Programming:**
  These are programs that teach a user something, which may be factual, e.g. about a geographical region, or a period of history, or some branch of chemistry or biology, or which develop a practical or intellectual skill, such as typing, spelling, doing arithmetic, doing logic, speaking grammatically, understanding geometry, etc. This can also overlap with other categories, especially gamey programming (since many games are educational) and "thinky" programming since an intelligent tutoring program does not merely go through a list of tasks presented to the learner but may also need to respond to questions from the learner or select or construct tasks designed to match the attainments and misunderstandings detected in the learner.

  A chess program that automatically adjusts its level of play so as to challenge and educate its opponent (which no current program can do, as far as I know) would be a thinky-teachy program!

- **Thinky programming (most relevant to educating educators)**

  Usually, the aim of a “Thinky” program is not to produce a working system whose behaviour is useful or entertaining, or which helps users understand some piece of science or mathematics, but to help learners think about ways of getting machines to do things humans and other animals do when they perceive, learn, solve problems, achieve goals, make and execute plans, solve puzzles, communicate in sentences, compose poems or music, or engage in competitive games where winning is not a matter of being lucky, big, fast, or physically skilled, but requires use of intellectual powers.

  Much of what these programs do, like thinking in humans, is invisible, though the results of the internal processing may be made partly visible by testing the programs. For example, a program that has learnt a new language may be able to answer questions posed in that language. A program that has learned to understand diagrams, may be able to produce descriptions of diagrams or may be able to create a diagram matching a description, such as “A square containing a circle above a triangle and part of a horizontal line sticking out to the left of the square”.

  Often it is very difficult to design a “thinky” program to perform a task without first doing a lot of exploratory programming to discover design possibilities and their consequences.

  This is why AI languages offer support for optimising human exploration by programming rather than emphasising support for optimising some end product to be put on the market (which would sometimes be written in a different language from the one used for exploration and development).

  Use of list-processing as a general mode of representation of structured information, rather than the more common tools of applied computing, can often facilitate very rapid and flexible exploration of a class of algorithms, because of strong support for run-time interactive modification of code and structures, while exploring the domain.

Using type-free list structures facilitates discovery and deployment of powerful abstractions.
Strongly typed languages assume the discoveries have been made before programming begins.
That's why I would rather teach beginners Pop-11 than Java.

Many of the hardest programming errors to identify and fix are of a "semantic" variety that would not be detected by syntactic checking: the program runs without crashing, but does the wrong thing, because the problem was not understood.

So strong support for run-time, interactive, debugging at a high level is more important than strong compile time checking – especially when that restricts expressive power of programs.

Examples of "thinky" programming tasks of varying difficulty are available at
`http://tinyurl.com/thinky-ex`.

Some are particularly simple, because they are meant to be suitable for absolute beginners.
Other examples were once regarded as suitable for PhD projects in the early days of AI, though with recent AI programming tools the programs are much simpler to create than they were originally.

A type of thinky programming that has a long history in AI is programming a machine to write programs, otherwise known as "automatic programming". A type of example that might be used in the classroom could be a program that takes in a verbal description of a task and then generates code to perform that task: e.g. "Draw a triangle with a square on the left and a circle enclosing both", or "Draw a cat in a box with the body of the cat hidden by the sides of the box". There are many more possibilities.

An alternative would be a program to produce a verbal description when given a drawing, where the description is not just a label, like "cup", or "horse", but specifies parts of the scene and various kinds of relationships between the parts, e.g. "is inside", "touches", "overlaps with", "too big for", "supports", "obstructs", etc.

There are infinitely many different types of thinky programming.
What sort of mind can explore that space?

We must foster educational diversity, not regimentation

Many teachers, exam boards, politicians, and industry advisers seem not to understand the educational need to feed into most, or perhaps all, our pre-university learners (starting very young) experience of computational thinking supported by practical programming experience in languages supporting learning, with much diversity across schools and learners.

This diversity is important because of the need to have a wide range of types of knowledge and experience about forms of computation in the culture at large.

There is unfortunately a wide-spread myth that there are a few “great ideas” about computation that everyone should learn – which ignores the many great computational ideas invented or discovered by evolution, including many we have not yet noticed?

If we focus education on too small a subset of the possible topics that can be taught effectively between the ages of about five and eighteen, we impoverish the “gene pool” of ideas available to the nation for many different purposes for decades to come.

NEWSFLASH

See the recent announcement (Sept 2012) that many portions of DNA that were previously facetiously(?) labelled “junk DNA” are actually concerned with high level control functions in the development of an individual from a fertilised egg, including, for example, turning other portions of the system on or off, on the basis of information available at various stages during the development. [Ref??]
Contrast an operating system’s booting scripts.

Giving learners opportunities: an extreme case

TED talk on youtube by Sugata Mitra

http://www.youtube.com/watch?v=dk60sYrU2RU

"Indian education scientist Sugata Mitra tackles one of the greatest problems of education – the best teachers and schools don't exist where they're needed most.

In a series of real-life experiments from New Delhi to South Africa to Italy, he gave kids self-supervised access to the web and saw results that could revolutionize how we think about teaching."

Suggestion:

The main function of educators is providing opportunities to learn.

I.e. not filling jobs, helping the economy, ...

The variety of possible opportunities and learnable contents is constantly changing as a result of changes (sometimes advances, but sometimes fashions) in science, mathematics, technology, music, other art forms, types of business, types of social interaction, natural problems, human-made problems ... and liberation from old, restrictive, cultural and religious constraints on ways of thinking (mind-binding).

Unfortunately, for many children, actions of parents, teachers, peers, religious communities, and marketing companies, introduce serious obstacles to learning, either because those in control lack relevant knowledge and abilities, or because they have oppressive or even bigoted attitudes and beliefs.

("Mind binding" is at least as bad as "foot-binding" practised in some communities).

Learning requires an information-processing architecture

Educators need to understand that different information-processing architectures, with different mechanisms, different forms of representation, different environmental challenges, ... support different forms of learning.

- A child's information-processing architecture expands in complexity and functionality.

  Including increasing parallelism of function: reading is a highly parallel process. (Why?)

- We understand very little of this, and much of the research into what does and doesn't work in education has no basis in deep explanatory computational theory.
- Getting beyond alchemy requires developing deep theories of learning and development, comparable to the ways physics and chemistry provided deep theories explaining the phenomena discovered empirically by alchemists.

  Studying physics and chemistry includes trying to create new machines and chemical substances to find out what does and does not work, which sometimes helps us understand natural phenomena.

  Likewise, trying to build learning machines can help us understand processes of learning and development (but not easily, or quickly).

  (Contrary to common opinions, there's much in common between how birds fly and how many aeroplanes fly: e.g. common principles of aerodynamics, including tradeoffs between stability and manouverability, and changing requirements for lift and thrust.)

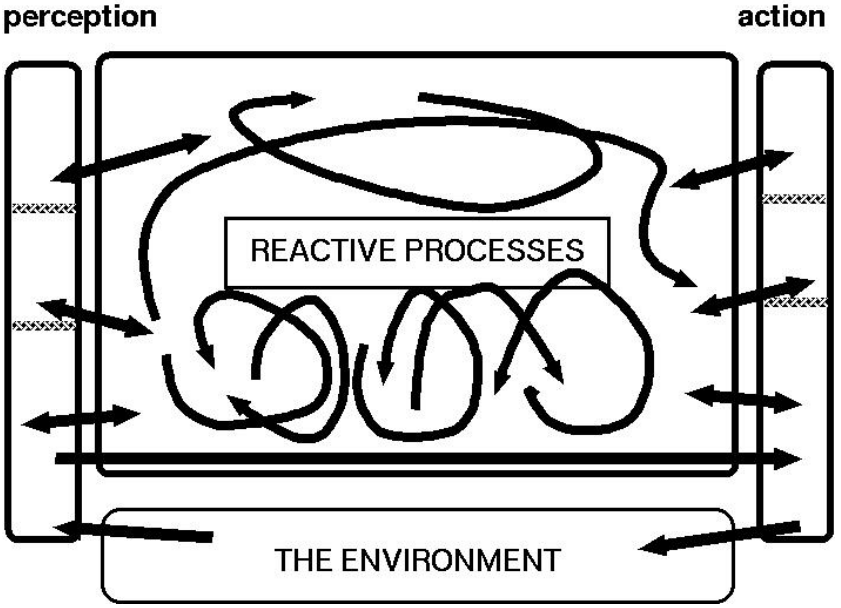
- What does a deep theory of education, or learning look like?

  It should refer to information-processing architectures, mechanisms, forms of representation, ontologies, mechanisms for self monitoring and self extension, ...
  Nature/nurture tradeoffs are important but not understood (Chappell & Sloman, 2007)

Compare: A simple (insect-like) architecture

A reactive system does not construct complex descriptions of possible futures, evaluate them and then choose one – it simply reacts: internally or externally.

(But see proto-deliberation, later.)



Several reactive sub-mechanisms may operate in parallel.

Processing may use a mixture of analog and discrete mechanisms.

An adaptive system with reactive mechanisms can be a very successful biological machine.

Some purely reactive species also have a social architecture, e.g. ants, termites, and other insects.

Are they all purely reactive? Can some deliberate? Compare the Portia spider. (Tarsitano, 2006)

Purely reactive biological species are precocial: they have large amounts of genetically determined capabilities, though minor environmentally driven adaptations are possible.

There are various ways reactive mechanisms can learn or adapt: changing links, changing strengths of links, growing new options in an array of possibilities, altering the environment (unintentionally) to guide future behaviours, ...

MAIN Features of reactive organisms

The main feature of reactive systems (in the normal AI sense of "reactive") is that they lack the core ability of deliberative systems, namely

they cannot represent and reason about phenomena that either do not exist or are not sensed, but might possibly exist.

E.g.:

future possible actions or events,
remote entities, or inaccessible interiors of perceived things,
the past, hidden items, etc.

i.e. they lack the ability to handle fictions and mere possibilities

- In principle, a reactive system can produce any external behaviour that more sophisticated systems can produce (e.g. using huge collections of pre-computed condition-action rules, including some internal conditions and actions)
- However, in practice there are constraints ruling this out, for instance the need for physical memories too large to fit on a planet.
- Evolution produced "fully deliberative" mechanisms in a subset of species, which overcame those and other constraints. (Sloman, 2006)
- Note:

  Deliberative mechanisms have to be *implemented* in reactive mechanisms, in order to work: but that does not stop them having deliberative capabilities.

Education over-emphasising skills treats learners as trainable reactive insects.

We'll see later what else they may be.

PROTO-DELIBERATIVE SYSTEMS

Evolution also produced proto-deliberative species:

- In a reactive system (e.g. implemented as a neural net) some sensed states with mixtures of features can simultaneously activate two or more incompatible response-tendencies (e.g. fight and flee).
- In that case some sort of competitive mechanism can select one of the options, e.g. based on the relative strengths of the two sensory patterns, or possibly based on the current context (internal or external e.g. level of hunger or whether an escape route is perceived).

  Here alternative futures are represented and then a selection is made in a context-sensitive way.

  Some people use "deliberation" to include such conflict resolution. E.g. (Arbib, 2003)

  However, such a system lacks most of the features of a fully deliberative system so we can call it a proto-deliberative system.

Going beyond reactive or proto-deliberative systems towards fully deliberative systems requires major changes in the architecture, though evolution may have got there by a collection of smaller, discrete, changes: **we need to understand the intermediate steps.**

Note: 'deliberative' and 'symbolic' are not synonyms. A purely reactive system may use symbolic condition-action rules (e.g. Nilsson's 'teleoreactive systems' (Nilsson, 1994)).

Sometimes the ability to plan is useful

Deliberative mechanisms

These provide the ability to represent unsensed possibilities

(e.g. possible actions, possible explanations for what is perceived, possible states of affairs behind closed doors, possible futures, possible histories).

One consequence is ability to plan multi-step actions, including nested actions

(unlike 'proto-deliberation', which considers only alternative single-step actions, and can use simple neural net mechanisms).

Another application could be building an interpretation of what's being read, or heard:

Compare Johnson-Laird on the role of models in comprehension of language.
`http://www.amazon.co.uk/Mental-Models-P-N-Johnson-Laird/product-reviews/0521273919/ref=sr_1_1_cm_cr_acr_txt`

Much, but not all, early symbolic AI (some of it surveyed in Margaret Boden's *Artificial Intelligence and Natural Man* (Boden, 1978)) was concerned with deliberative systems (planners, problem-solvers, parsers, theorem-provers, concept-learners, analogy mechanisms), in simple architectures.....

There were also experiments with reactive systems: e.g. simple simulated creatures that reacted to their needs, drives, and externally sensed phenomena, and possibly learnt in simple ways.

To illustrate, there are demo movies of a purely reactive symbolic simulated sheepdog herding sheep, and a hybrid deliberative/reactive one, with planning capabilities here:

`http://www.cs.bham.ac.uk/research/poplog/figs/simagent/`

Varieties of deliberative mechanisms

Deliberative mechanisms differ in various ways:

- the forms of representations (often data-structures in virtual machines)
- the variety of forms of representation available together
  (e.g. logical, pictorial, rules, vectors of values, network weights, ...)
- the algorithms/mechanisms available for manipulating representations
- the kinds of 'compositional semantics' available,
  e.g. Fregean (function application), analogical (picture composition), hybrid forms, etc.
- the depth of 'look-ahead' in planning (depth-first/breadth first)
- the number of possibilities that can be represented simultaneously and compared
- the ability to chunk (discretise) possibilities, so as to allow multi-step planning.
- the ability to represent future, past, or remote present objects or events
- the ability to represent possible actions of other agents
- the ability to represent mental states of oneself or others
  ('meta-semantic' competences linked to meta-management, below).
- the ability to represent abstract entities (numbers, rules, proofs, plans, errors, ...)
- the ability to learn, in various ways,
  including developing new formalisms, new ontologies, new forms of inference, ...

Most deliberative capabilities require the ability to learn and use new abstract associations, e.g.

- between situations and possible actions
- between actions and possible effects
- between spatial structures/relationships and constraints on possibilities.

What should educators know about architectures?

Which biological information processing mechanisms support deliberative capabilities?

What roles do the various mechanisms and architectures play in learning and development?

How much of the architecture is there at birth, and how do new components grow?

Some suggestions about complex interactions between nature and nurture, and how new architectural layers may be grown in ways that depend on what had previously been grown, which may depend in various ways on the environment, are presented in (Chappell & Sloman, 2007)

In such cases the computational challenges in design of DNA are formidable: how can manipulations of chemical structures produce the ability to learn any one of several thousand human languages?

In what ways can such developmental processes vary between individuals?

In what ways can the development of information-processing mechanisms, formalisms, and architectures vary (or go wrong?) and what do good teachers need to understand about that?

Some, but not all of these topics are developed below.

Fully deliberative systems need Meta-Management

Symbolic AI up to the mid 1980s mainly addressed tasks for which deliberative systems were appropriate.

But only a small subset of deliberative mechanisms was explored, and the processing architectures were not well designed for systems performing most tasks humans can do — e.g. they lacked meta-management abilities to monitor and control deliberation.

G.J. Sussman's HACKER (Sussman, 1975) was one of the few exceptions.

Some progress was made towards a class of systems with 'fully deliberative' capabilities, including:

- The ability to represent what does not yet exist, or has not been perceived.
- The ability to use representations of varying structure
  – using compositional semantics supporting novelty, creativity, etc.
- The ability to use representations of potentially unbounded complexity
  (Compare fixed size vector representations, e.g. in neural nets.)
- The ability to build representations of alternative possibilities, compare them, then select one.

Recently, more AI researchers have added reflective and meta-management capabilities, using meta-semantic capabilities

Including the ability to monitor, detect, categorise, evaluate, plan, debug internal processes, including deliberative processes. (See Marvin Minsky *The Emotion Machine* (Minsky, 2006).)

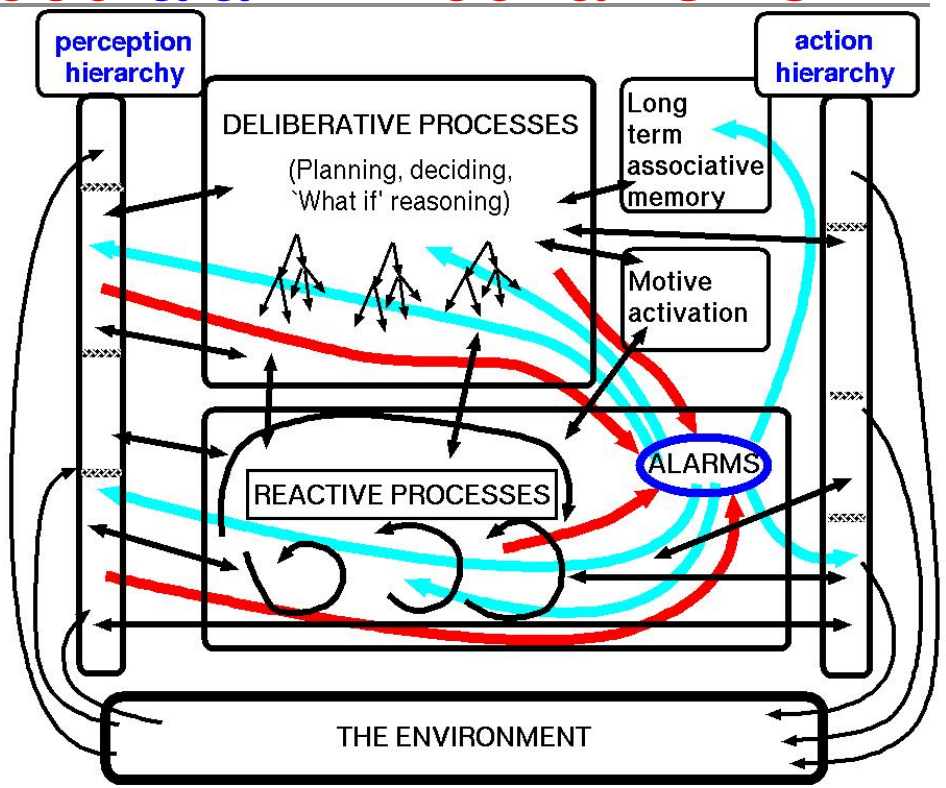
Intelligent systems need alarm mechanisms

NB: my diagrams are "impressionistic", not to be treated as design specifications.

In unpredictable environments, reactive alarm mechanisms with interrupt and redirection capabilities are useful in any architecture — whether purely reactive, deliberative, fully deliberative, etc.

A reactive alarm system can override, interrupt, abort, or modulate processing in other systems.

Inputs to alarm mechanisms may come from anywhere in the system, and outputs may go to anywhere in the system.



Alarm subsystems can make mistakes if they use fast rather than careful decision making, so how they are trained can be crucial.

Learning can both extend the variety of situations in which alarms are triggered and improve appropriateness or accuracy of alarm responses.

False positive alarms and false negatives can result both from limitations in the learning mechanism and from features of the individual's history: illustrated by many aspects of human emotion.

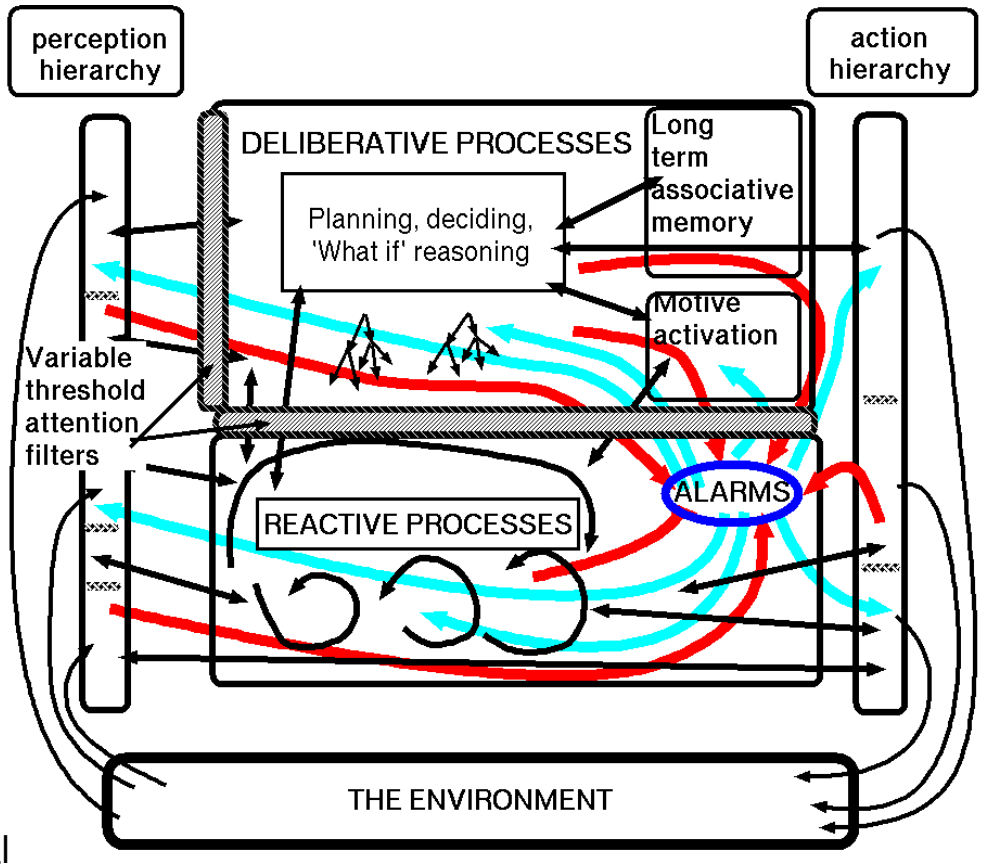
Even reading fictional stories can activate such alarm mechanisms!

And bad teaching?

Some alarms may need filtering

Alarm signals produced by unintelligent reactive mechanisms could disrupt more urgent and important deliberative process.

Attention filters (depicted crudely here) with dynamically modulated thresholds, can suppress some alarms and disturbances during urgent and important tasks.



Many human emotions involve perturbances and limitations of attention filtering mechanisms, including some long term emotions, like grief:

I.P. Wright, A. Sloman & L.P. Beaudoin, (1996), Towards a Design-Based Analysis of Emotional Episodes, *Philosophy Psychiatry and Psychology*.

(Wright, Sloman, & Beaudoin, 1996) `http://tinyurl.com/BhamCog/96-99.html#2`

(Much of the paper was inspired by real life reports of human grief, especially long term grief – which doesn't fit most theories of emotions.)

Much learning in mathematics, programming and other deep fields involves developing new "alarm" mechanisms able to detect potentially bad moves at early stages. (Sussman: compiling new critics.) More generally, meta-management requires meta-semantic competences (next slide).

More layers of abstraction, and meta-management

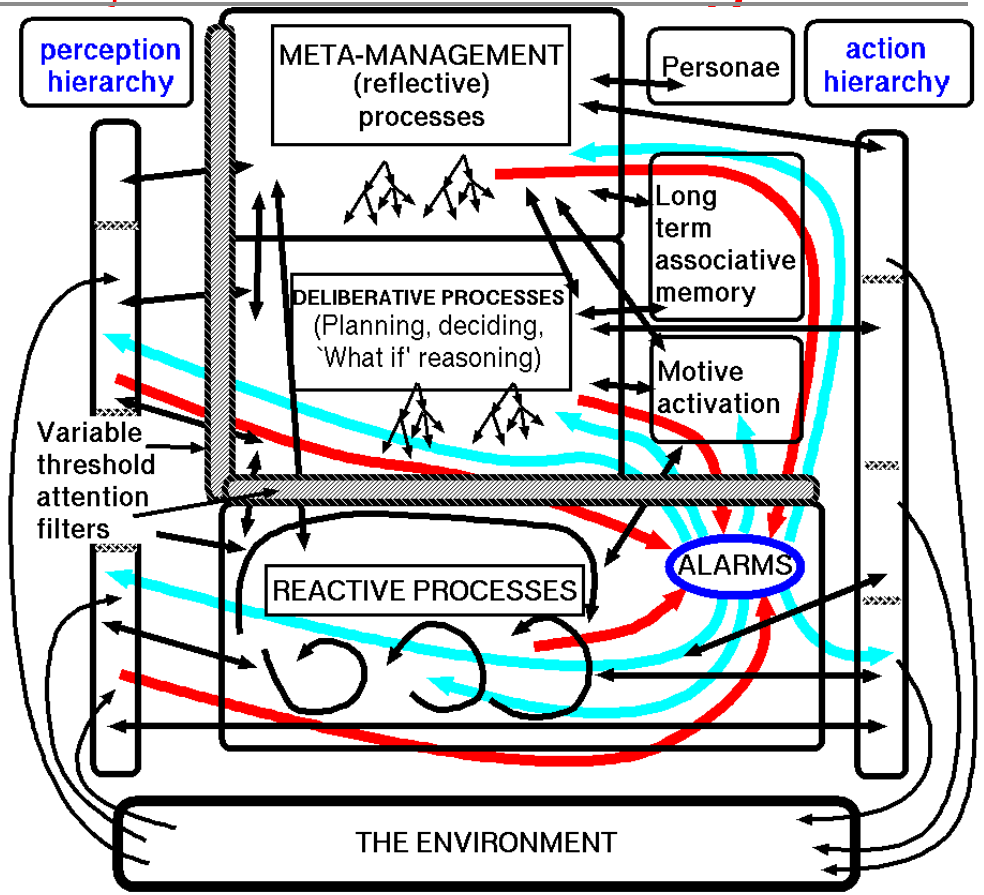
H-Cogaff: a conjectured type of architecture (Perhaps in future robots.)

Arrows crudely represent a few information flow paths (including control signals, reports, questions, goals, interrupts, modulation...)

High level control regimes need to vary, with different “personae” active in different contexts.

If meta-management processes have access to intermediate perceptual databases, this can produce self-monitoring of sensory contents, leading robots with such architectures to discover sensory qualia.

Some will then become robot philosophers.



Meta-management systems need to use meta-semantic ontologies: they need the ability to refer to things that refer to things.

Exactly what is required depends on the environment and individual histories.

(Compare developmental psychology research on “Mind reading” in children, e.g. (Apperly, 2010))

There’s lots more on these topics in CogAff papers and presentations:

`http://tinyurl.com/BhamCog/#overview`

`http://tinyurl.com/BhamCog/talks`

Two questions about architectures, for educators

If one of the most important processes in education and development is extension and modification of the information processing architecture, don't educators need to understand architectures and how they change and grow?

Conjecture

The kind of "training" of children (skill development) that aims to produce good results in tests that feed into league tables for schools, can develop reactive sub-modules at the expense of other parts of an information processing architecture.

Another conjecture

Poorly designed educational programmes may emphasise and build on forms of motivation that are **reward-based**, ignoring the much more powerful and subtle forms of motivation that are **architecture-based**, and drive deeper forms of learning

See "Architecture-Based Motivation vs Reward-Based Motivation" (Sloman, 2009) online here: `http://tinyurl.com/BhamCog/09.html#907`

Of course, both rewards and development of skills, including reactive skills, can be very important.

I am only warning about getting the emphasis wrong –
especially for learners with the highest intellectual potential.

Biological pressures towards self-knowledge, self-evaluation and self-control

A deliberative system can easily get stuck in loops or repeat the same unsuccessful attempt to solve a sub-problem, or use thinking strategies with flaws.

- One way to reduce this is to have a parallel sub-system monitoring and evaluating the deliberative processes.

  (Compare Minsky on "B-brains" and "C-brains" in *Society of Mind* (Minsky, 1987))

- We call this meta-management (following Luc Beaudoin's 1994 PhD thesis).
  (Beaudoin, 1994) Online: `http://tinyurl.com/BhamCog/81-95.html#38`,

  It seems to be rare in biological organisms and probably evolved very late –
  (in altricial species? (Chappell & Sloman, 2007))

- As with deliberative and reactive mechanisms, there are many forms of meta-management.

I suspect human mathematical competences grew out of meta-management competences that allowed important features of observed situations (including possibilities and constraints) to be detected, and abstracted for future use in varied situations.

How many teachers can explicitly, actively, encourage and support that kind of learning and development? Could more be helped by computational thinking?

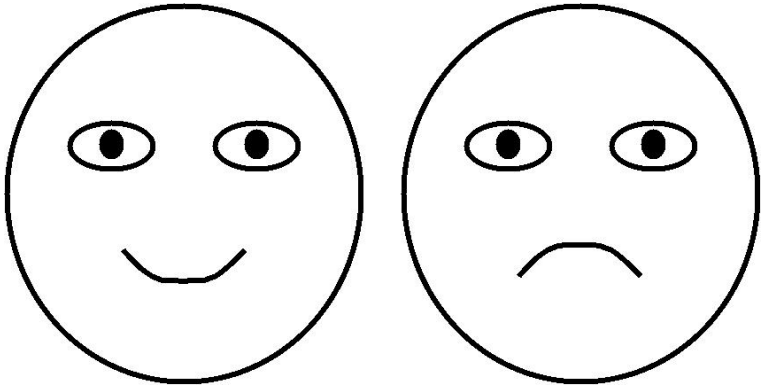
Inner and outer perception co-evolved

Conjecture:

Some of the representational meta-management capabilities that evolved for dealing with **self-**categorisation can also be used for **other-**categorisation, and vice-versa.

Perceptual mechanisms may have evolved recently to use those representational capabilities in percepts: perceiving not just physical structures and behaviours, but also mental states:

Example: seeing someone else as happy, or angry, or trying to do X.



This is an example of "multi-window" perception, summarised below.

Do you see the eyes in the right face as different from the eyes in the left face?

Are they geometrically different?

What might this tell us about human perceptual information processing?

Multi-window perception and action

If multiple levels and types of perceptual processing go on in parallel, we can talk about

"multi-window perception",

(Different kinds and scales of information taken in simultaneously.)

as opposed to

"peephole" perception.

(Information available is in a restricted form and only a small amount can be accessed at a time.

In multi-window perception, perceptual processes operate concurrently at different levels of abstraction serving the needs of different cognitive processing layers.

Likewise, in an architecture there can be multi-window action or merely peephole action.

Conjecture: developmental brain abnormalities can limit "multi-window" capabilities.

Compare current robots:

Much work on robotics ignores the need for multi-window perception and multi-window action in intelligent animals and machines.

Many AI perceptual mechanisms operate only at a single level on a single task, e.g. tracking something moving. Likewise the action mechanisms.

Animals that evolved to cope with environments in which many processes go on in parallel relevant to each individual's needs, opportunities, risks, and constraints, must have architectures that can process different kinds of information in parallel, for different purposes.

What are the implications for educators?

Contrast teaching dance, sporting activities, musical sight-reading, mathematics, poetry, acting ....

Where does language fit in?

Clearly language is crucial to humans.

It is part of the process of cultural transmission that accelerates changes in competence, and within individuals it extends cognitive capabilities in many ways (e.g. being able to think about what would have happened yesterday if the weather had not been so bad, and being able to do science and mathematics).

Equally clearly many animals lacking human language have considerable intelligence, shown in hunting, building nests in trees, guarding and caring for offspring, social relationships, tool-making etc.

Pre-linguistic human children have many kinds of competence.

CONJECTURE:

In order to understand (and replicate) human linguistic competence we need to understand the architectures that suffice for other intelligent species and pre-verbal children, and work out how such architectures might grow to support linguistic abilities.

It will very likely involve extensions of different kinds in perceptual mechanisms, in all the central processing layers, and in the motor sub-systems.

Mechanisms that proved powerful for development in other altricial species may be crucial for human language learning.

What is their role in learning to read (and write)? Do educators need to know?

Some Implications

Within this framework we can explain (or predict) many phenomena, some of them part of everyday experience and some discovered by scientists:

- Several varieties of emotions: at least three distinct types related to the three layers: primary (exclusively reactive), secondary (partly deliberative) and tertiary emotions (including disruption of meta-management) – some shared with other animals, some unique to humans. (For more on this see Cogaff Project papers)
- Discovery of different visual pathways, since there are many routes for visual information to be used. (See `http://tinyurl.com/BhamCog/talks/#talk8`)
- Many possible types of brain damage and their effects, e.g. frontal-lobe damage interfering with meta-management (Damasio).
- Blindsight (damage to some meta-management access routes prevents self-knowledge about intact (reactive?) visual processes.)

This helps to enrich the analyses of concepts produced by philosophers, scientists and engineers sitting in their arm chairs.

Without attempting (and failing) to build working systems, it is very hard to dream up all the requirements, and all the varieties of architectures, mechanisms, forms of representation, states, and processes that can occur or are needed, if you merely use your own imagination, as philosophers and some others try to do.

Without all that, we can't understand the varieties of learning and development.

Implications continued ....

- Many varieties of learning and development
  (E.g. “skill compilation” when repeated actions at deliberative levels train reactive systems to produce fast fluent actions, and action sequences. Needs spare capacity in reactive mechanisms, (e.g. the cerebellum?). We can also analyse development of the architecture in infancy, including development of personality as the architecture grows.)
- Conjecture: mathematical development depends on development of meta-management – the ability to attend to and reflect on thought processes and their structure, e.g. noticing features of your own counting operations, or features of geometrical and topological constraints in your visual processes.
- Further work may help us understand some of the evolutionary trade-offs in developing these systems.
  (Deliberative and meta-management mechanisms can be very expensive, and require a food pyramid to support them.)
- Discovery by philosophers of sensory ‘qualia’. We can see how philosophical thoughts (and confusions) about consciousness are inevitable in intelligent systems with partial self-knowledge.

For more see papers and presentations here:

`http://www.cs.bham.ac.uk/research/cogaff/`
`http://www.cs.bham.ac.uk/research/projects/cogaff/talks/`

Conclusions?

Too many to summarise here:

Computational thinking can profoundly transform our understanding of ourselves and the world we inhabit (which includes many, increasingly many, kinds of natural and artificial computational systems).

We need to educate our youngsters to learn far more varieties of computational thinking than any of their ancestors – and more than current computer scientists and IT engineers.

The variety of types of computational thinking needed to understand ourselves and our world is so great that limiting the syllabus to perceived **current** needs can be dangerously counter-productive.

Allow creative teachers and learners to explore many directions – using powerful tools to support the exploration, including “thinky programming”, which is unnecessarily difficult with most currently popular programming languages.

Don’t let the tail of assessment wag the dog of learning.

And make the use of league tables for educational institutions **illegal!**

Thanks for the invitation!

And apologies to the ALT-C organisers for messy slides and and chaotic presentation.

I have too many debts to acknowledge, especially

Immanuel Kant, Gottlob Frege, Noam Chomsky, Jean Piaget, John Holt, Max Clowes, Marvin Minsky, Seymour Papert, John McCarthy, Herbert Simon, Annette Karmiloff-Smith,

and many colleagues and former colleagues and students at Sussex University and Birmingham University (UK, not Alabama)

and contributors to poplog including members of ISL, and the usenet pop-forum and poplog-dev members,

and many children who have stretched my mind,

and probably key people I've temporarily forgotten to mention.

Google: "Aaron Sloman" presentations, for these and more online slide presentations, and information about the meta-morphogenesis project.

A new experimental site: `https://sites.google.com/site/slomanmetamorphogenesis/`

Some references – lots more needed

References

Apperly, I. (2010). *Mindreaders: The Cognitive Basis of "Theory of Mind"*. London: Psychology Press.

Arbib, M. A. (2003). Rana computatrix to Human Language: Towards a Computational Neuroethology of Language Evolution. *Philosophical Transactions: Mathematical, Physical and Engineering Sciences,*, *361*(1811), 2345–2379. (http://www.jstor.org/stable/3559127)

Beaudoin, L. (1994). *Goal processing in autonomous agents*. Unpublished doctoral dissertation, School of Computer Science, The University of Birmingham, Birmingham, UK. Available from `http://www.cs.bham.ac.uk/research/projects/cogaff/81-95.html#38`

Boden, M. A. (1978). *Artificial intelligence and natural man*. Hassocks, Sussex: Harvester Press. (Second edition 1986. MIT Press)

Chappell, J., & Sloman, A. (2007). Natural and artificial meta-configured altricial information-processing systems. *International Journal of Unconventional Computing*, *3*(3), 211–239. (http://www.cs.bham.ac.uk/research/projects/cosy/papers/#tr0609)

Craik, K. (1943). *The nature of explanation*. London, New York: Cambridge University Press.

Gibson, J. J. (1979). *The ecological approach to visual perception*. Boston, MA: Houghton Mifflin.

Minsky, M. L. (1987). *The society of mind*. London: William Heinemann Ltd.

Minsky, M. L. (2006). *The Emotion Machine*. New York: Pantheon.

Nilsson, N. (1994). Teleo-reactive programs for agent control. *Journal of Artificial Intelligence Research*, *1*, 139–158.

Sloman, A. (1978). *The computer revolution in philosophy*. Hassocks, Sussex: Harvester Press (and Humanities Press). Available from `http://www.cs.bham.ac.uk/research/cogaff/crp`

Sloman, A. (1984a). Beginners need powerful systems. In M. Yazdani (Ed.), *New horizons in educational computing* (pp. 220–234). Chichester: Ellis Horwood Series In Artificial Intelligence. (http://www.cs.bham.ac.uk/research/projects/cogaff/81-95.html#45)

Sloman, A. (1984b). Experiencing Computation: A Tribute to Max Clowes. In M. Yazdani (Ed.), *New horizons in educational computing* (pp. 207 – 219). Chichester: Ellis Horwood Series In Artificial Intelligence. (http://www.cs.bham.ac.uk/research/projects/cogaff/00-02.html#71)

Sloman, A. (2006, May). *Requirements for a Fully Deliberative Architecture (Or component of an architecture)* (Research Note No. COSY-DP-0604). Birmingham, UK: School of Computer Science, University of Birmingham. Available from `http://www.cs.bham.ac.uk/research/projects/cosy/papers/#dp0604`

Sloman, A. (2009). Architecture-Based Motivation vs Reward-Based Motivation. *Newsletter on Philosophy and Computers*, *09*(1), 10–13. Available from `http://www.apaonline.org/publications/newsletters/v09n1_Computers_06.aspx`

Sloman, A. (2012, June). *Is education research a form of alchemy?* (Vol. 27). Available from `http://tinyurl.com/BhamCog/misc/alchemy/`

Sussman, G. (1975). *A computational model of skill acquisition*. San Francisco, CA: American Elsevier. Available from `http://dspace.mit.edu/handle/1721.1/6894`

Tarsitano, M. (2006, December). Route selection by a jumping spider (Portia labiata) during the locomotory phase of a detour. *Animal Behaviour*, *72, Issue 6*, 1437–1442. Available from `http://dx.doi.org/10.1016/j.anbehav.2006.05.007`

Turing, A. M. (1952). The Chemical Basis Of Morphogenesis. *Phil. Trans. R. Soc. London B 237*, *237*, 37–72.

Wing, J. M. (2006, Mar). Computational Thinking. *CACM*, *49*(3), 33–35. Available from `http://www.cs.cmu.edu/afs/cs/usr/wing/www/publications/Wing06.pdf`

Wright, I., Sloman, A., & Beaudoin, L. (1996). Towards a design-based analysis of emotional episodes. *Philosophy Psychiatry and Psychology*, *3*(2), 101–126. Available from `http://www.cs.bham.ac.uk/research/projects/cogaff/96-99.html#2`