ARTIFICIAL INTELLIGENCE DEVELOPMENT ENVIRONMENTS

Aaron Sloman
http://www.cs.bham.ac.uk/˜axs/
School of Computer Science
The University of Birmingham

How AI programming is both like and unlike other forms of software engineering and how this influences design of AI languages and tools.

Talk to first year AI students, University of Birmingham, December 2001. 2002
Accessible as talk 11 here
http://www.cs.bham.ac.uk/research/cogaff/misc/talks/
(Talk 10 is 'What is Artificial intelligence.')

WHAT IS ARTIFICIAL INTELLIGENCE?

It is more general than some definitions imply:

AI is a (relatively) new approach to some very old problems about the nature of mind and intelligence.

It combines with and contributes to several other disciplines, including:

- psychology,
- philosophy,
- linguistics,
- biology,
- anthropology,
- logic,
- mathematics,
- computer science & software engineering,

and other subjects that study humans and other animals.

AI is not a branch of Computer Science, nor a purely engineering discipline.

AI has two main kinds of goals

- **Science**

  i.e. studying things that already exist or might exist, explaining how they work, searching for general principles relevant to understanding

  - people,
  - other animals,
  - intelligent machines of the future,
  - and perhaps creatures from other parts of the universe.

- **Engineering**

  using that knowledge to solve practical problems, including

  - making new useful kinds of machines,
  - producing new forms of entertainment
  - perhaps helping us manage ourselves better, e.g. in education, therapy, ...
    - because we understand ourselves better
    - because we have new tools.

See also http://www.cs.bham.ac.uk/˜axs/misc/aiforschools.html

AI and computer science

AI uses computer science, just as physics uses mathematics, but AI is not computer science, just as physics is not mathematics.

AI uses computers because they are the best available tool, not because they are the object of study.

If we knew now to make brains we would sometimes use those instead of computers, for some of the sub-tasks of modelling or explaining intelligence.

Already there are attempts to build neural networks that are partly like brains, and do not work like ordinary computers.

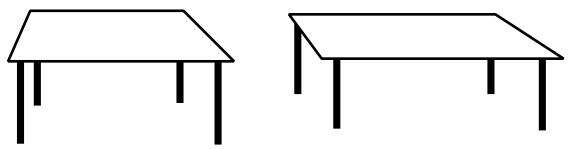
But brain science is not computer science.

ARE BRAINS COMPUTERS?

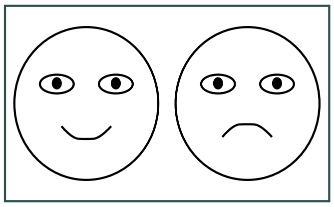
Examples of research problems

Vision – the hardest problem in AI

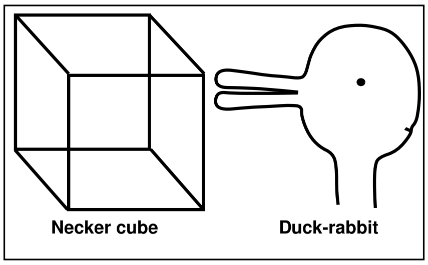
How do we get from 2-D patterns of illumination on our retinas to percepts of a 3-D world?



How do we see expressions of emotion in faces?



How can we see the same 2-D visual input in different ways?

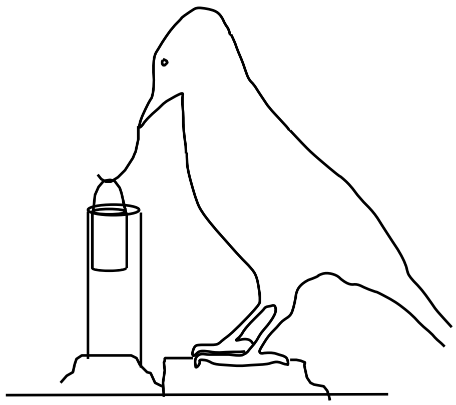


And many more including perception of motion, visual pleasure, etc.

AI is also concerned with language, learning, reasoning, action control, etc., etc.

Betty Crow: Cognitive Agent and Hook-maker

Two crows, Betty and Abel, learnt to use bent wire to fish a bucket of food out of the vertical tube (as in the picture). Then Abel flew off with the hook.



See the video here: http://news.bbc.co.uk/1/hi/sci/tech/2178920.stm
To find more, give google: betty crow hook

- Betty tried using a straight piece of wire for a while, and failed.
- She then pushed one end of the wire into the tape holding the tube and moved the other round with her beak, making a hook, which she used to lift the bucket.
- She did this 9 times out of 10. Reported in Nature and shown on BBC TV (August 2002).

COULD A ROBOT REPLICATE BETTY'S MENTAL PROCESSES?

What sort of architecture could do what Betty did?

Many explanations are compatible with any observed performance, e.g.:

- Pure chance?
- An innate behaviour triggered by some mixture of internal and external state?
  - What mixture?
  - How did the genes get the information? Why was it selected?
- A learnt adaptation in a trainable (altricial) reactive system?
  - What sort of boot-strapping could achieve this?
  - How is the learnt information acquired, represented, stored, activated, used?
- Was it a deliberative (e.g. problem-solving) process?
  - Using what sort of ontology for possible goals, states, actions?
  - Using what general knowledge?
  - Invoked how?
  - Acquired how? (Using an architecture built in infancy?)
  - Using what planning mechanisms? (Using what representations, what search mechanisms?)
- Did it involve self-knowledge? (Reflection/meta-management)
  - Did Betty understand what she was doing, or did she, like many AI deliberative systems, lack reflection/meta-management? (Can a crow teach another crow to do this?)

The questions are deep and important because understanding of spatio-temporal processes can be re-used in many contexts.

E.g. doing mathematics, designing architectures, thinking about anything complex.

Vision and affordances

Vision is not just about

- Object recognition
- Perception of geometrical and physical structure and motion
- Building cognitive maps for route-planning

There's something deeper, not yet properly characterised, which can be called perception of affordances.

The idea comes from the psychologist, J.J.Gibson (1979) *The Ecological Approach to Visual Perception*

- Affordances are not "objective" properties intrinsic to physical configurations.
- They are relational features dependent on the perceiver's
  - Common or likely goals and needs
  - Capabilities for action (physical design + software)
  - Constraints and preferences (avoid stress, injury)

What affordances did Betty need to see?

What sort of computer-based system (robot) could see them, and use them to solve similar problems?

How can you solve the vision problems?

We don't know precisely what a vision system does, so we cannot just specify what it has to do and then derive a design for it.

- You won't necessarily find out what vision is by reading psychological books on vision either: since mostly the people who write them have never designed a system (e.g. a robot) that uses its visual mechanisms to do something. Trying to build such a working system helps you understand why vision is needed, and what it has to do. (The requirements are different for different animals. Why?)
  See talk 8 on vision here: http://www.cs.bham.ac.uk/~axs/misc/talks/
- The process of interweaving theory, modelling, testing, experimenting, can go through many cycles over a long period of time, and involves several disciplines, including robotics, psychology, brain science.
- AI contributes by developing new explanatory constructs (forms of representation, algorithms, information processing architectures) which can contribute to the theories. And tools to help the process.

> Initial ideas are nearly always wrong: they just don't work, except in very limited contexts. So we need tools for exploring, testing, de-bugging and extending half-baked ideas.

Common to science and engineering aims: Building WORKING Models

The scientific and engineering objectives of AI are *both* served by building working models (computer programs, robots, etc.)

– to test our theories
or
– to perform useful tasks

But usually we don't start by knowing what sort of program we are aiming to produce: AI programming is generally *exploratory*.

In the course of developing the program you discover

- ➔ What sort of environment the system has to interact with
- ➔ What sort of information the program will acquire
- ➔ How that information needs to be represented and stored
- ➔ How the information needs to be accessed and manipulated
- ➔ What sorts of internal and external behaviours are required
- ➔ What sort of system architecture is needed

In contrast, Software Engineering, or engineering in general, often (not always) starts with a well-defined objective.

Typical software engineering "life-cycle"

→ a) Specify high level objectives
→ b) Produce more detailed functional specification
→ c) Produce high level design for system
→ d) Produce more detailed designs
→ e) Implement designs
I.e. write code (using editor)

→ f) Compile programs
→ g) Link programs
→ h) Test programs
→ i) Run debugger
Often involves examining core dump
→ k) Go back to one of (a–e)

→ l) Sell or deliver system
→ m) Users find bugs and design flaws
→ n) Investigate problems
→ o) Go back to one of (a–e)

Tools for standard software engineering

The following tools are used for different parts of the preceding software engineering life-cycle:

- Specification formalism
- Programming language
- Editor
- Compiler

  Needs to be able to do as much checking of program at compile time as possible, to save debugging later, and to optimise code.
- Linker
- Executable user program

  (Produced by compiler and linker from output of editor + libraries)
- Debugger

NB: the above are usually separate tools run in sequence.

- Others, e.g. project management tools, program verifiers.
- Brains, users, customers, ...

An unusual function of AI programming

In AI research, programming is often part of the process of discovering what the problem is, not only finding a solution.

E.g.:

– Understanding what the environment is really like
– Understanding what the perceptual problems are
– Understanding how the users see, learn, feel
– Discovering what will please the users
– Finding out what sorts of linguistic constructs the users can understand, or are likely to produce
– Finding out what sort of behaviour makes an intelligent tutoring system
– Finding out what information about the environment a robot’s visual system needs to provide.

Vision is not always just pattern recognition, or object recognition, or distance estimation, or...

Typical AI investigation “life-cycle”

- a) Specify high level objectives: often very vague.
- b) Find out what is already known about the problem (often not much) and form hunches about the rest. Find and read relevant literature!
- c) Select suitable AI development environment (toolkit, programming language, etc.) and try to sketch a partial solution, to get something that can run.
- d) Implement the partial solution, testing bits as you go. I.e. write code (using integrated editor), test it, try various modifications, all while the program is running. Get help from end-users to check things, where appropriate.

  To deal with obscure bugs or problems write new exploratory code, e.g. testing contents of data-structures, changing them to see what happens, etc.

  (Note: the whole system, including program under test, *is* the debugger.)

  Use results to refine problem, theory and code. Occasionally throw all code away and start again.

- Sell or deliver system, or publish papers.
  Make models available for others to use and develop.
- Find another problem domain or spend the rest of your life revising or extending the theory and the system.

AI programming uses many different techniques

Often an AI problem, e.g. designing a robot, or an intelligent tutoring system, requires combining a wide variety of types of information manipulated in a wide variety of different ways.

Including:

- Symbol-manipulating programs
- Logical reasoning systems
- Neural nets
- Genetic algorithms (evolutionary computation)
- Dynamical systems (based on models from physics)
- Where appropriate, new sorts of hardware

Do not believe anyone who tells you that only one kind of technique works: some ignorant people have narrow-minded views of AI

But we can't tell in advance what sort of formalism is going to be useful. So AI languages (especially Lisp and Pop-11) have very powerful means of *extending* the language, including extensions at run time, because the compiler is part of the running system, not a separate tool.

Some characteristics of AI languages

- Often use incremental compilers or interpreters
  Having compiler available at run time gives enormous extra power.
- The symbol table is also available at run time,
  So variables are objects of the program, not just entities used at compile time.
- Wide variety of built-in data types (often including lists)
- Ability to define new derived data types
- No (or little) compile time type-checking
- Rich run-time type-checking facilities
  e.g. procedures like isword, isnumber, islist, isproperty
- Automatic garbage collection
  Previously unique to AI systems. Now available in Java also: non-AI language designers are learning from AI.
- Extendable: allowing use of sub-formalisms that suit sub-problems:
  I.e. the syntax of the language is not fixed, which gives great flexibility, but makes it harder to develop automatic program analysis and checking tools.
- Often support mixed language programming
  e.g. Pop-11 or Lisp + Prolog

Characteristics of AI languages continued

- May have integrated editor (written in an AI language)

  Allows editor to be used as a system interface tool, e.g. Ved/XVed
- No separate debugger: instead users develop debugging tools to suit their applications.

  E.g. programs to interrogate suspect data-structures when things go wrong.
- Errors usually don't abort, but save useful state and allow resumption.

  E.g. Mishap in Pop-11
- May include support for concurrency

  (E.g. lightweight processes in pop-11)
- May include support for object oriented programming

  (Classes, sub-classes, inheritance, methods, often with more power than non-AI object oriented languages, like Java. See Objectclass in Pop-11 and CLOS in Lisp)
- Allow development of shareable, re-usable packages that extend the language, and can be added at run time when needed.

  These can extend library search lists and documentation directories.

  E.g. Poprulebase, Rclib in Pop-11.

Characteristics of AI languages continued

- Procedures are often 'first class objects'

  (A running program can refer to procedures, store them in data-structures (e.g. lists, arrays) can create them, etc. Providing support for various kinds of "closures" – using partial application or lexical closures.)
- The program can manipulate itself (e.g. write and compile, or interpret source code, and create new procedures at run time.).
- Some AI languages support systems that need knowledge of their own run-time system state, e.g. what is on the control stack, etc.

EXAMPLE: POPLOG
An AI development environment

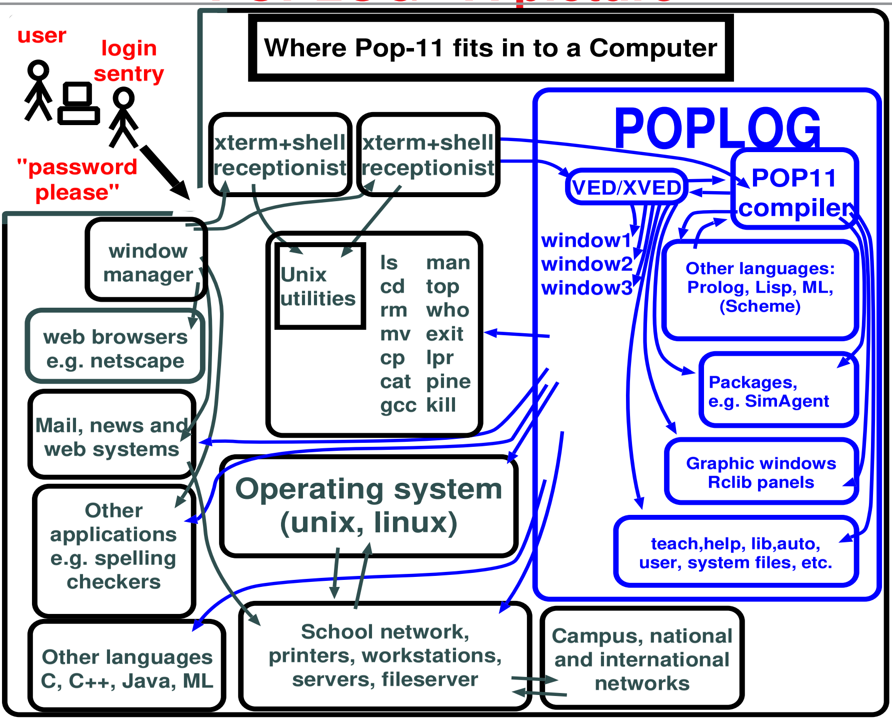
- The Poplog system, used for teaching and research in AI in Birmingham (and other places) is described here

  http://www.cs.bham.ac.uk/research/poplog/poplog.info.html
- It used to be an expensive commercial product but is now available free of charge with full system sources, by courtesy of Sussex University and SPSS (formerly Integral Solutions Ltd.)
- It is still used in a number of commercial packages, but often commercial developers who use Pop-11 or Poplog do not advertise the fact.
- An example of a very successful commercial product developed mainly using Pop-11 is Clementine.

  See http://www.spss.com/clementine

The next slide is a diagram, giving a rough (partial) indication of the internal structure of Poplog and some of its connections with other parts of a typical computing system.

POPLOG - A picture



See also: http://www.cs.bham.ac.uk/~axs/cogaff/simagent.html

Other AI development systems

There are many other powerful AI development environments and toolkits, often based on Lisp or Prolog, e.g.

– Franz Lisp
– Sicstus Prolog
– Others described here

http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/air.html

These may also be useful:

http://www.aaai.org/AITopics/aitopics.html
http://www.cs.bham.ac.uk/research/cogaff/misc/talks/#talk10
http://www.iit.nrc.ca/misc.html

Also use htp://www.google.com and search for “AI resources”

MAYBE YOU CAN DESIGN SOME NEW DEVELOPMENT TOOLS FOR AI IN THE FUTURE?