Session 1b: Programming paradigms and A.I. overview

Some of the major ways of expressing logic for a computer.

With particular attention to the **functional** paradigm

COMP 378 -- Spring 2017
Mr. Weisert

The **procedural** paradigm

- Most common way of expressing computer logic.
- Central to major programming languages, especially older ones:
  - Fortran
  - Algol
  - Cobol
  - PL/I
  - Pascal
  - C
  - BASIC

- Mimics the way the computer hardware itself works (The **stored program** concept)
- Also arises as an (escape) option within many languages that emphasize a different paradigm

*What other programming paradigms are there?*

---

### Programming paradigms

**Q:** Which two are important to A.I.?

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Early example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural</td>
<td>Algol</td>
</tr>
<tr>
<td>Object-oriented</td>
<td>Smalltalk</td>
</tr>
<tr>
<td>Relational</td>
<td>SQL</td>
</tr>
<tr>
<td>Functional</td>
<td>LISP</td>
</tr>
<tr>
<td>Logical</td>
<td>Prolog</td>
</tr>
<tr>
<td>Spreadsheet</td>
<td>VisiCalc</td>
</tr>
<tr>
<td>Visual</td>
<td>CA-Realizer</td>
</tr>
<tr>
<td>Report generation</td>
<td>RPG</td>
</tr>
</tbody>
</table>

### Important paradigms for A.I.

<table>
<thead>
<tr>
<th>Paradigm</th>
<th>Early example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Procedural</td>
<td>Algol</td>
</tr>
<tr>
<td>Object-oriented</td>
<td>Smalltalk</td>
</tr>
<tr>
<td>Relational</td>
<td>SQL</td>
</tr>
<tr>
<td><strong>Functional</strong></td>
<td>LISP</td>
</tr>
<tr>
<td>Logical</td>
<td>Prolog</td>
</tr>
<tr>
<td>Spreadsheet</td>
<td>VisiCalc</td>
</tr>
<tr>
<td>Visual</td>
<td>CA-Realizer</td>
</tr>
<tr>
<td>Report generation</td>
<td>RPG</td>
</tr>
</tbody>
</table>
Paradigms and popular programming languages for A.I. (inference engines)

- **Forward chaining** logic: (functional paradigm)
  - LISP (ca. 1959)
    - Scheme
    - Common LISP
    - Clojure
  - OPS5

- **Backward chaining** logic: (logical paradigm)
  - PROLOG (ca. 1972)

The functional paradigm

- A program is a collection of functions, some of which invoke other functions (i.e. a hierarchy).
- Execution consists of evaluating the top-level function or expression.

  That sounds awfully simple!

  What’s the catch?

Question

- C, a fundamentally procedural language, has one construct that supports the functional paradigm. What is it?
- C++, Java, and C# inherit that construct.

(Answer later)

Example

- C has no operator for exponentiation, $x^n$.
- Therefore, to raise a numeric value $x$ to a power $n$, we must apply a named function.
  
  \[ \ldots \text{power}(x,n) \ldots \]

- Standard library functions are available.
  - But $\text{pow}(x,y)$ in the C# or Java Math pseudo-class is too general for many uses.
  - The power (exponent) $y$ is floating point!
  - We need a simpler version just for integer powers.

  Why do we call Math a pseudo-class?
Naive version #1

static double power(double x, int n)
returns(double)
{
    double result = 1.0;
    for (k=1; k <= n; ++k)
        result *= x;
    return result;
}

Will that work?
What about negative n?
Other comments?

Naive version #2

static double power(double x, int n)
returns(double)
{
    double result = 1.0;
    double expr = x;
    int pwr = n;
    if (n < 0)
    {
        expr = 1.0 / x;
        pwr = -n;
    }
    for (k=1; k <= pwr; ++k)
        result *= expr;
    return result;
}

Will that work?
Is it reasonably efficient?

Efficiency?

Q: How many multiplications will the previous version perform in order to compute \(x^{36}\)?

A: 35 (!)  Is that reasonable?
What's a better way?

A much more efficient approach

Here's a possible succession of operations and results for \(x^{36}\):

- Initialize result to 1.0, then:
  - \(\text{result} \times x\)
  - \(\text{result} \times \text{result} \times x^2\)
  - \(\text{result} \times \text{result} \times \text{result} \times x^4\)
  - \(\text{result} \times \text{result} \times \text{result} \times \text{result} \times x^8\)
  - \(\text{result} \times \text{result} \times \text{result} \times \text{result} \times \text{result} \times x^{16}\)
  - \(\text{result} \times \text{result} \times \text{result} \times \text{result} \times \text{result} \times \text{result} \times x^{32}\)

Only seven multiplications!
Integer powers

- To compute \( x^n \)
  
  - \( x^{2n} = (x^n)^2 \) (even exponent)
  
  - \( x^{2n+1} = (x^{2n}) \times x \) (odd exponent)

Recursive approach

- Instead of thinking of the computational sequence:
  
  \[ x, x^2, x^4, x^8, x^9, x^{18}, x^{36} \]

- Think of the top-down (recursive) sequence:
  
  \[ x^{36}, x^{18}, x^9, x^8, x^7, x^2, x \]

Odd case

Recursive version #1
(pure procedural)

```java
public static double power(double x, int n)
    returns(double)  {
if (n == 0) return 1.0;    // Base
double tempo     = power(x, n/2); // Even
if (n < 0) return power(1.0/x,-n); // Negative
if (n == 1) return x;       // cases
if (n % 2 == 1) return x * power(x,n-1); // Odd
    return tempo * tempo;    // power
}
```

How about that? Compare with powers on previous slide?

Final version

- There are lots of ways to improve the previous version in minor ways.

- But one interesting way is to apply the functional paradigm!

  How can we do that?

  in C?    (assignment #1)
  in Clojure?
Earlier question revisited

- C, a fundamentally procedural language, has one construct that supports the **functional paradigm**.

  What is it?

- C++, Java, and C# inherit that construct.

Answer

- It's the `?:` operator
  - Reduces logic to **evaluating an expression** rather than performing a sequence of steps.

- Compare with LISP or Clojure **cond**
  
  https://clojuredocs.org/clojure.core/cond

- The executable part of `power(x, n)` can be a single `return` statement!

  *Should it be? Is this an improvement?*

LISP and the functional paradigm

- LISP stands for "list processing"

- **Everything** is a list
  - data
  - function definitions
  - function invocations

- Function notation
  - The function name is *inside* the list, so instead of `fctn(arg1, arg2, arg3)` we code `fctn arg1 arg2 arg3`  
  - Instead of infix notation for operators, we use prefix, not `a + b` but `(+ a b)`

  *What's the advantage of that?*

Surprise

- Despite its use in many of the most advanced and sophisticated computer applications, LISP is one of the **oldest** programming languages!

  - Devised ca. 1958 by John McCarthy (MIT)
  
  http://www-formal.stanford.edu/jmc/history/lisp/lisp.html

  - You should find that interesting, but you won't be responsible for details on our examinations.
Evolution of Lisp

- Improvements were made over the years, but versions diverged, it was hard to get rid of awkward obsolete constructs.
- ANSI standard Common Lisp (1994)
- CLOS (Common Lisp Object System) included object-oriented concepts

Functions in LISP (Clojure, too)

- A function can create and invoke other functions! That can be extremely powerful.
- Recursion is assumed
- Multi-threading is assumed
- Clojure allows comma separators (for clarity), but experienced functional programmers rarely use them.

Artificial Intelligence

a traditional definition

- Computerized logic that mimics behavior (perception, deduction, decision making, etc.) commonly thought to require human intelligence.

What's wrong with that?

How can we assess whether a computer exhibits such behavior?

Problems with the traditional definition

- Computerized logic that mimics behavior (perception, deduction, decision making, etc.) commonly thought to require human intelligence.

Who does that common thinking?

The boundary keeps moving.
### The Turing test
- Can a human being interacting or conversing with an unseen partner accurately judge whether that partner is a person or a machine?
- If not, a machine passes the test.

http://en.wikipedia.org/wiki/Turing_test

### Some topics in A.I.
- Expert systems
  - inference, deduction
  - problem diagnosis and advice
- Computer vision and other senses
- Natural language understanding
  - automated translation
  - speech synthesis
- Neural networks
  - machine learning

http://en.wikipedia.org/wiki/Neural_Networks_(journal)