Overview

- Announcement: GCL and other Lisp interpreter options.
- Finish up leftovers from last lecture.
- How to find a research topic (in AI).
- Assignment #1: page 13.
- Lisp (if time permits).

Announcement

- GCL is actually available in:
  /pub/www/faculty/daugher/gcl/gcl
- On some older machines, it may be found in:
  /usr/local/bin/gcl
- There is also a commercial version of Common Lisp which is free to students:
  - Allegro Common Lisp
  - Supports Linux, windows, FreeBSD, Mac OS X

Finding a Research Topic in AI

1. Broadly survey the field of AI
   - Browse through the table of contents of textbooks, etc.
   - Reading various online tutorials is immensely helpful.
2. Identify the problem that you want to solve. Evaluate if it is a significant problem.
3. Find out related literature and critically read them.
4. Find out related books and browse them.
5. Identify gap of knowledge.
6. Think about an idea to fill that knowledge.
7. Go ahead and test your idea.

Gathering the Raw Material

You cannot generate a new idea if your brain is empty:

- Turn your web-browsing skills into productive use.
- Sources of information:
  - Books: single/multi author, monographs, and edited volumes.
  - Handbooks and encyclopedias.
  - Papers: journals, conference proceedings, tech reports.
  - Web search: www.researchindex.com
Once you start reading papers, you will get to know several authors. When reading other papers, check the references for these names. This way you can figure out what’s going on.

Read the conclusion (you may or may not prefer this) to get a more concrete grasp on the theme.

Try to remember the authors (at least the first author).

Read the discussion to find out what are the immediate or more long-term issues in the field. Identify viable research directions.

Find out more about your favorite authors on the web. Fully exploit journals that are accessible online through the TAMU library (they have institutional subscriptions).

Don’t read further until you know what is the main theme.

When reading a paper, read the abstract, and read the conclusion to grasp the overall message of the paper.

Read the introduction if you’re still interested after that.

Read the discussion to identify weaknesses of the paper and how it relates to other work.

For single author books, read the book reviews. We don’t have infinite time!

There are several very good independent reviewers:

– dannyreviews.com
– www.santafe.edu/shalizi/reviews/

Quick view of current trends: what is the bleeding edge in this field?

May need to be selective in picking a paper to read.
Handbooks and Encyclopedias

These are edited volumes of short (1–2 pages) tutorials or introductions on specialized topics.

- Very convenient and easy to access.
- Quickly gain broader perspectives on a specific area.
- Identify the most relevant references, etc.
- Look for online encyclopedias, and annotated bibliographies.

Critical Reading Is Important

- Remember: don’t just read, read critically.

Finding a Problem

- Look into other paper’s discussion.
- Read review papers.
- For short-term projects, choose a small but significant problem: this is the focus of the current course.
- For long-term projects, choose one or more related problems that can together solve a complex problem.

Forming an Idea

Traditional method:

- Gather raw material: reading, etc. Make notes (this is very important).
- Think about the solution, and how to relate the raw material.
- For a couple of days or several days, shove away the whole thing and totally stop thinking about the subject consciously.
- Idea will flow.

I would add, actively explore, e.g. on the web. Idea will flow faster!
Assignment #1

- Browse the table of content of the textbook.
- Browse my web page: faculty.cs.tamu.edu/choe, especially the Resources section.
- Find a subject that interests you.
- Submit a written (or typed) statement on paper including:
  1. description of the subject,
  2. why you find it interesting and important, and
  3. what are the main issues or hot problems.
- Read and revise the short summary several times.
- Submit it before class on Monday 9/9. Include references.

LISP: A Quick Overview

- Components: Atoms, Lists, and Functions.
- Basics: list, math, etc.
- Arrays and SETQ vs. SETF
- Variable binding
- Lexical vs. dynamic scope
- Conditionals, predicates, iterations, etc.
- User-defined function
- Recursion
- Output

Some Potential Topics

These are very broadly defined:

- Analogy
- Active Agents
- Selective Attention
- Causal Inference
- Bayesian Inference
- Symbol Grounding
- Computational Perception: Vision, Audition, etc.
- and much more.

LISP: A Quick Overview

Components

Symbolic expression = ATOM or LIST.

- Atom:
  \[
  [<\text{letters}>]<\text{digits}>+ \\
  \text{e.g.: } 1, 10, \text{foo, bar, this-is-an-atom}
  \]

- List:
  \[
  "(" [<\text{list}>]<\text{atom}>"+ ")" \\
  \text{e.g.: } (a), (1 (1 2 3) (4 5 6))
  \]

- NIL: it is an atom and at the same time a list. NIL is the same as ()

- T: true, as opposed to NIL. See conditionals and predicates.
Basics

- quote: returns a literal (i.e. not evaluated) atom or a list.
  \[ (+ 2 3) \rightarrow (+ 2 3) \]
  \[ (quote (+ 2 3)) \rightarrow (+ 2 3) \]
  Compare with:
  \[ (+ 2 3) \rightarrow 5 \]
  \[ (eval '(+ 2 3)) \rightarrow 5 \]
- first, second, third, ..., nth, rest:
  \[ (first '(a b)) \rightarrow A \]
  \[ (nth 2 '(a b c d)) \rightarrow B \]

Basics: List

- car: returns first element (atom or list)
  \[ (car '(a (b c))) \rightarrow A \]
  \[ (car '((b c) a)) \rightarrow (B C) \]
- cdr: returns all except the first element of a list, as a list
  \[ (cdr '(a (b c))) \rightarrow ((B C)) \]
  \[ (cdr '((b c) a)) \rightarrow (A) \]
- Combinations are possible: `cXXXr` where `X=(a|r)`
  \[ (cdr '(a (b c))) == (car (cdr '(a (b c)))) \rightarrow (B C) \]
- list: creates a list out of atoms and lists
  \[ (list 'a '(1 2) '((3 5) (7 8))) \rightarrow (A (1 2) ((3 5) (7 8))) \]
- length: number of elements in a list
  \[ (length '(a b c)) \rightarrow 3 \]

Basics: Assignments/Arrays

- setq: assignment of value to a symbol
  \[ (setq x 10) \rightarrow 10 \]
  \[ x \rightarrow 10 \]
- setf: can set the value of a symbol (== setq) or location or structure (next slide).

Arrays and SETQ vs. SETF

- make-array: create an array
- aref: array reference
- setf: set value of array element
Arrays and SETQ vs. SETF

```lisp
> (setq a (make-array '(3 3)))
#2A((NIL NIL NIL) (NIL NIL NIL) (NIL NIL NIL))
> (aref a 2 2)
NIL
> (setf (aref a 2 2) 1000)
1000
> a
#2A((NIL NIL NIL) (NIL NIL NIL) (NIL NIL 1000))
> (setq (aref a 2 2) 1000)
Error: (AREF A 2 ...) is not a symbol.
```

Basics: Math

- (+ 1 2) (* 3 4) (+ (* 2 3) (/ 4 5)) etc.
- (max 1 2 3 4 5) (min 4 6 5)
- (sqrt 16) (expt 2 3) (round 3.141592)

Basics: File Loading

- (load "filename")

Group of Commands

`progn` returns the result of the last element, but evaluates all s-expressions in the argument list.

- `(progn (setq a 123) (* 5 10))` → 50
  a → 123

Binding

You can bind variables anywhere in a program with the `let` or `let*` special forms to create a local context.

- `let` and `let*`: lexical scope (local context)
  (let (local var list) BODY)
  (let ((x 10) y (z 20)) BODY)
  (let* ((x 10) (y (* 2 x)) z) BODY)
- Either just a variable or (variable default-value).
- With `let*`, values from previous vars can be used to define new value.
  (let* ((x 10) (y (* 2 x)) z) BODY)
Binding: Example

> (let ((a 3)) (+ a 1))
4
> (let ((a 2))
    (b 3)
    (c 0))
    (setq c (+ a b))
  c
5
> (setq c 4)
4
> (let ((c 5)) c)
5
> c
4

Lexical Scope

Return value according to the lexical scope where it was defined.

> (setq regular 5)
5
> (defun check-regular () regular)
CHECK-REGULAR
> (check-regular)
5
> (let ((regular 6)) (check-regular))
5

Dynamic Scope

Use the defvar to define a special variable that is dynamically scoped.

> (defvar *special* 5)
*SPECIAL*
> (defun check-special () *special*)
CHECK-SPECIAL
> (check-special) 5
> (let ((*special* 6)) (check-special)) 6
> *special* 5
> (let ((x 23)) (check-special)) 5

Conditionals: the Ps.

p is for predicate:

- numberp, listp, symbolp, zerop, ...
- common comparisons: <, >,
- equal: if the values are the same.
- eq: if the memory locations are the same.
- and, or, not: logical operators.

Returns either NIL or T.
Control Flow

IF STATEMENT
(if (> 2 3) ; condition
  (+ 4 5) ; when true
  (* 4 5) ; when false
)

SWITCH STATEMENT
(cond ((testp1 (return-value1)) ; condition 1
  ((testp2 (return-value2)) ; condition 2
    ((testp3 (return-value3)) ; condition 3
      (t (default-value)) ; default
    

Function

- defun: user defined function
  > (defun mult (x y) (* x y) )
  DEFUN
  > (mult 10 20)
  200

- Use the let and let* forms:
  (defun mult (x y)
    (let ((tx x) (ty y))
      (* tx ty)
    )
  )

Recursion

- Fibonacci number:
  F(N) = F(N-1) + F(N-2), F(1)=1, F(2)=2.
  (defun fibo (x)
    (cond
      ((equal x 1) 1)
      ((equal x 2) 2)
      ((> x 2)
        (+ (fibo (- x 1)) (fibo (- x 2))))
    )
  )
  >(fibo 4)
  5
  >(fibo 5)
  8

Iterations

DOTIMES
(dotimes (index-var upper-bound result-var) BODY)

> (dotimes (k 1 val) (setq val k))
0

> (dotimes (k 10 val) (setq val k))
9

Also find out more about dolist, do, and loop.
• print: print a string
  (print "hello")

• format: format a string; (format dest string args)
  ~% : insert CR
  ~S : S-expression
  ~A : ascii
  ~D : integer
  ~widthD : blank space e.g. ~5D
  ~F : floating point
  ~width,decimalF : width and decimal point

  Output

  Format: examples

  > (format t "Hello, world!")
  Hello, world!
  NIL

  > (format nil "Hello, world!")
  "Hello, world!"

  Format: examples

  > (format nil "The list is ~s and~%the text is ~a"
     (list 'a 'b 'c)
     "This is a string"
   )
  "The list is (A B C) and
  the text is This is a string"

  Format: examples

  > (format nil "One: ~d~%Two:~f~%Three:~5,2f"
     12 (/ 4 3) (/ 4 3)
   )
  "One: 12
  Two:1.3333334
  Three: 1.33"
Dealing with Errors

>aa <-- error

Error: The variable AA is unbound.
Fast links are on: do (si::use-fast-links nil) for
Error signalled by EVAL.
Broken at EVAL. Type :H for Help.
>>:h <-- get help
...
>>:q <-- go back to top level

Top level.
>