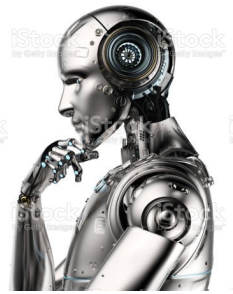
CSC 244/444
Machine Reasoning: Introduction



Instructor: Len Schubert (WH 3003, sometimes on Zoom, 394 165 2563)
TAs: Yifan Zhu (grad), Qianqian Wei (undergrad)

Course focus: *Reasoning and planning (in arbitrary domains)*

Suppose I have 3 cubical blocks, A, B, and C, and currently
B and C are directly on the table, and A is on C.
How can I build a stack where C is on top, B is in the middle,
and A is at the bottom, by moving one block at a time?

Currently, Large Language Models (LLMs) like ChatGPT are all the rage.
Can ChatGPT solve this? It answered incorrectly, building a stack with
A on the top, B in the middle, and C at the bottom! It flubs many variants.

Another small combinatorial reasoning problem:

In the recent election of a student council president at Brighton High School, the turnout was 100% -- every Brighton student voted for some candidate. Clyde and Diane were two of the Brighton students who were candidates in the election. (Our own kids at Brighton, Alice and Bob, were not candidates.) Sadly, every candidate in the election disparaged every other candidate. Of course, no-one votes for someone they disparage.

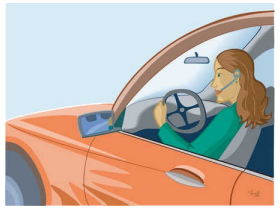
As briefly as you can, answer these questions: Did any of Clyde, Diane, Alice, and Bob vote for themselves? If so, who, and what is your reasoning?

ChatGPT answered that no students voted for themselves! When reminded that the candidates only disparaged *other* candidates, not themselves, ChatGPT decided there wasn't enough information to draw a conclusion!

Some examples used in this course *before* the advent of LLMs:

Alicia drove home from her holiday visit to the US.

What border(s) did she cross in her drive?



Posing this to ChatGPT led to a long exchange, with ChatGPT saying it didn't have enough information. It did mention Canada and the US as neighbors at some point, but even after I said her visit was to the "Lower 48", ChatGPT thought Alicia might be from US territories like Puerto Rico (it's hard to drive on water!). When I ruled that out, CharGPT still pleaded insufficient information.

Truman Everts (1870) was lost in Yellowstone in the winter for 37 days; he had nothing but a small knife and opera glasses.

How could he survive?

I posed this to ChatGPT (but altering the name & date), and GPT did fairly well, though not as quite well as Everts. BTW it knows about Everts, but hallucinates about how he survived. LLMs have good *general* knowledge but invent specifics.

Course goals and logistics

Please see LKS's course web page at
https://www.cs.rochester.edu/u/schubert/444/
Check out the links, as well …

You should understand

- Course goals: methods of representing factual knowledge, using it for inference and planning; readiness for KR&R research; basics in symbolic programming
- Requirements
- Text (for 444), materials
- Resources for learning Lisp
- Schedule, lecture-by-lecture (and % weights)
- Supplementary readings (Steven Pinker – intelligence; Asilomar AI principles – beneficial AI; Paul Kennedy – Malthus redux; AI & Life in 2030 – what (not) to expect, how to proceed; Kai-Fu Lee –the *real* threat of AI)

"KR&R" in AI

What are "logical" representations of knowledge?

- Formalization of language, enabling "mechanical" reasoning of various kinds;
- "About-ness": symbols & expressions refer to things/properties in the world (*denotational semantics*); sentences (in language & logic) can be true or false;
- In general, we need to add *specialized* (symbolic & analog) representations;

Kinds of reasoning and planning:

- Deduction (but *not* exclusively!)
- Abduction, induction (uncertain)
- By analogy, and use of familiar patterns

Sources of knowledge

- Perception
- Physical interaction
- linguistic interaction, listening, reading;
  great unsolved problem: ("few-shot") learning via language ("KA bottleneck")

Current status of AI, Future of AI

Achievements

- ChatGPT, LLaMa, PaLM, Bard, … (pretrained on trillions of words, using many billions of parameters) but "black box", and no trustworthy reasoning, planning;
- Deep Blue, Alpha Go/Zero, AlphaFold, Midjourney, "self-driving" cars, MT, …
- Siri, Alexa, Google Assistant, Cortana, …,
- Jeopardy! (IBM's Watson), Wolfram|Alpha, …
- Rule-based systems for system design, banking, geological prospecting, trouble-shooting, diagnosis, call routing, inventory maintenance, …
- Proofs of difficult mathematical results, Prolog type-checking for Java VM, …

Outlook:

- Symbolic/DNN (Transformer, LLM) integration?!
- Truly self-driving vehicles, factory automation, construction, medical diagnosis/ treatment, bank tellers, customer service, telemarketers, stock and bond traders, paralegals, radiologists, home assistants, elder care, …
- "Singularity"? (I.J. Good – "intelligence explosion"; Vernor Vinge)
- Will we self-destruct first? (nuclear, climate, …) cf. Nick Bostrom re Fermi paradox

Richard Powers: We are

"pitched in a final footrace… between inventiveness and built-in insanity"

Common Lisp

"Hello world" ➔ prints this, upon hitting 'return'

Lisp is a functional language based on the λ-calculus;
e.g., (lambda (x) (expt n x)) is the same function as: $f(x) = n^x$ for all numeric x;
e.g., (lambda (n) (expt n x)) is the same function as: $f(n) = n^x$ for all numeric n;
e.g., (lambda (n x) (expt n x)) is the same function as: $f(n,x) = n^x$ for all numeric n, x;

In the first/second example, n/x must already have a pre-set numeric value,
e.g., (setq n 3), or (setq n 3.14159265).

UGs: sbcl (Steel Bank Common Lisp) or clisp
Exiting: (exit), or (quit)
stack list (after error): :down
local variables (after error): :list-locals
Debugging: http://www.sbcl.org/manual/#Debugger

Grads: sbcl or acl (/p/lisp/acl/linux/latest/alisp); In acl (for sbcl see notes):
Exiting: :ex
stack list (after error): :dn
local variables (after error): :loc
Debugging: https://franz.com/support/documentation/current/doc/debugging.htm

Common Lisp (some more comments)

We can give lambda-functions a name using 'defun', e.g.,
(defun exp-ratio (x) (/ (exp x) (expt 2 x))) ; the ratio $e^x/2^x$

So, e.g., (exp-ratio 2) → 1.847264; (exp-ratio (sqrt 2) → 1.54335

Arguments of functions in Lisp can be "anything":
- atomic symbols like JOHN, CSC244, |lower-case-symbol|, |Lower Case Symbol|, ...
- numbers like 3, 3.14, 17/5, ...
- strings like "Hello", "Nice to meet you", ...
- lists of any of the above, like (JOHN 3 "Hello")
- lists of atoms, lists, etc., like (CSC244 (JOHN 3 "Hello") (|Intel| (JOHN 3))), ...
- arrays, created via make-array and accessed via aref
- hash tables, created via make-hash-table and accessed via gethash
- structures (with keys and values), created via defstruct, make-, & setf and accessed via <struc>-<keyword> functions

Since data have the same (list structure) form as functions, we can write programs that create functions!

Common Lisp (some recursive list processing examples)

```
; Any list structure made up of atomic symbols can be viewed as a tree
; where only the terminal (leaf) nodes have labels -- namely the atoms
; in left-to-right order.
;
(defun preorder-leaves (tree)
; Function for returning tree (list structure) nodes in pre-order
  (cond ((null tree) nil)
        ((atom tree) (list tree))
        (t (append (preorder-leaves (car tree))
                   (preorder-leaves (cdr tree))))))

(setq tree '(A (B (C D) (E F)) ((G H) I)))

(preorder-leaves tree); ===> (A B C D E F G H I)
```

Common Lisp (some recursive list processing examples)

```
; Any list structure where all sublists start with atomic symbols can alternatively
; be viewed  as a tree where all nodes have labels, by regarding the first element
; of each sublist as a node label, and the remaining elements (if any) as the
; children of the node. .
;
; In this case, returning node labels in preorder is the same as applying
; preorder-leaves; i.e. this is the same as preorder-leaves:

(defun preorder-nodes (tree)
  (cond ((null tree) nil)
        ((atom tree) (list tree))
        (t (append (preorder-nodes car tree)) (preorder-nodes (cdr tree))))))
```

BTW, in writing code, use logical alignment of successive lines (even though Lisp doesn't care), and keep line lengths to about 75 – readability is important!

Common Lisp (some recursive list processing examples)

```
; Postorder is more interesting; i.e., we collect the postordered elements
; of the subtrees of a node before adding the root node (label). 'mapcar' can
; be used to postprocess all the subtrees (exclusive of the "root", i.e. first
; element) of a tree; we need to append the results (using (apply #'append ...))
; and then add the root to the end:

(defun postorder-nodes (tree)
; Function for returning tree nodes in post-order
  (cond ((null tree) nil)
          ((atom tree) (list tree))
         (t (append (apply #'append (mapcar #'postorder-nodes (cdr tree)))
                    (list (car tree))))
 )); end of postorder-nodes

(setq tree1 '(A (B (C D) (E F)) (G (H I J) K (L) ())))

(format t "~a" (postorder-nodes tree1))
; ===> (D C F E B I J H K L G A) ; NB: The '()' (i.e., NIL) element is ignored
```