CS 334
Lecture 3
Stephen Freund
Lisp

- Pure Functional language
- input → output
- No visible state

```
[ A ] → [ B / C ]
```

```
[ A ] → [ B ] → [ C ] → nil
```
Structured Data

1) Lists.

2) Trees

3) Structures.
Higher-Order Functions

- Functions that take a function as an arg.
- \((\text{mapcar } \text{'fn } (e_1,\ldots,e_n))\) => \((\text{fn}(e_1),\ldots,\text{fn}(e_n))\)
- \((\text{sort } l \text{'comp})\)

Lambda Function
Machine Model

- How language manages memory at runtime.

Garbage Collection

1. \((\text{car} \ (\text{cons} \ x \ y)) \Rightarrow x\)

   ![Diagram of garbage collection process]

   - delete cons cell
   - delete the cdr value.

2. \((\text{car} \ ((\lambda \ a \ \text{cons} \ a \ a) \ b))\)
Garbage - If P is running, the cons cell at location M is garbage if no contrived execution of P will touch M.

Garbage Collector
- Identify garbage & reclaim it.
- Part of runtime system, transparent to programmer.
Mark & Sweep Collector

- each cell has a tag.

1) set all tags to 0
2) set tag to 1 for all cells reachable from root locations (registers, function params, local vars, call stack).

Use graph reachability (OFS).

3) put all cells w/ tag 0 on a "free list".
Why is GC Good?

- Avoids memory errors.
- Matches functional style.
- Performance?
  - 30 years ago: $2^{-y} \times$ slowdown.
  - Now: < 20%
Side Effects

Pure Lisp: Evaluating expression doesn't change visible machine state.

\[
\begin{align*}
c & \colon a[i] \\
& \rightarrow a[i++] \\
& \text{have side effects.}
\end{align*}
\]
Impure Lisp

- Ok, it has side-effects.

- pair

- update list

  ① \( (\text{cons 'E (cdr list)}) \)
  ② \( (\text{replace list 'E}) \)

Which better?

② Allocates fewer cons cells.
① GC is harder of more sharing. (Kw)
① Side effects make code harder to reason about.