61A Lecture 29

Monday, November 7
Homework: Huffman Encoding Trees

Efficient encoding of strings as ones and zeros (bits).

A 0     C 1010     E 1100     G 1110
B 100    D 1011    F 1101     H 1111

Decoding a sequence of bits:

1 0 0 0 1 0 1 0
B A C
Logo Refresher

**Data types:** Words and sentences (immutable sequences)

**Syntactic forms:** Call expressions, literals, and to-statements

```logo
? (print sum 10 difference 7 3)
14

? run [print sum 1 2]
3

? to double :x
> output sum :x :x
> end

? print double 4
8
```
Logo Interpreter Architecture

Logo words are represented as Python strings

Logo sentences are represented as Python lists

The Parser creates nested sentences, but **does not** build full expression trees for nested call expressions
Tracking Positions in Lines

A line is used up as it is evaluated

A Buffer instance tracks how much of a line has been used up.

```python
>>> buf = Buffer(['show', '2'])
>>> buf.current
'show'
>>> print(buf)
['show', '2']
>>> buf.pop()
'show'
>>> print(buf)
['show', '2']
>>> buf.pop()
'2'
```
Evaluating Lines

Evaluating a line of Logo involves evaluating each expression

Evaluate a line

Evaluate the next expression

Calls repeatedly

? print 1 print 2
1
2

<table>
<thead>
<tr>
<th>logo_eval</th>
<th>Argument</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>first call</td>
<td>[ &gt;&gt; print, 1, print, 2 ]</td>
<td>prints 1, returns None</td>
</tr>
<tr>
<td>second call</td>
<td>[ print, 1 &gt;&gt; print, 2 ]</td>
<td>prints 2, returns None</td>
</tr>
</tbody>
</table>
The logo_eval function dispatches on expression form:

- A **primitive expression** is a word that can be interpreted as a number, True, or False. Primitives are self evaluating.
- A **variable** is looked up in the current environment.
- A **procedure definition** creates a new user-defined procedure.
- A **quoted expression** evaluates to the text of the quotation, which is a string without the preceding quote. Sentences are quoted and evaluate to themselves.
- A **call expression** is evaluated with apply_procedure.

```python
def logo_eval(line, env):
    """Evaluate the first expression in a line."""
    token = line.pop()
    if isprimitive(token):
        return token
    elif isvariable(token):
        ...
```
Evaluating Call Expressions

Apply a named procedure

apply_procedure

Return the output value

Evaluate \( n \) operands

collect_args

Return \( n \) arguments

Apply a procedure to a sequence of arguments

logo_apply

Return the output value

---

1. Collect 1 argument via logo_eval (collect_args)

2. Apply print procedure to the argument '2' (logo_apply)
class Procedure():
    def __init__(self, name, arg_count, body, isprimitive=False, needs_env=False, formal_params=None):
        self.name = name
        self.arg_count = arg_count
        self.body = body
        self.isprimitive = isprimitive
        self.needs_env = needs_env
        self.formal_params = formal_params

    def logo_apply(proc, args):
        """Apply a Logo procedure to a list of arguments.""
        if proc.isprimitive:
            return proc.body(*args)
        else:
            """Apply a user-defined procedure"""
Eval/Apply in Lisp 1.5

\[ \text{apply}[fn; x; a] = \]
\[ [\text{atom}[fn] \rightarrow [\text{eq}[fn; \text{CAR}] \rightarrow \text{caar}[x]]; \]
\[ \text{eq}[fn; \text{CDR}] \rightarrow \text{cdar}[x]; \]
\[ \text{eq}[fn; \text{CONS}] \rightarrow \text{cons}[\text{car}[x]; \text{cadr}[x]]; \]
\[ \text{eq}[fn; \text{ATOM}] \rightarrow \text{atom}[\text{car}[x]]; \]
\[ \text{eq}[fn; \text{EQ}] \rightarrow \text{eq}[\text{car}[x]; \text{cadr}[x]]; \]
\[ T \rightarrow \text{apply}[	ext{eval}[fn;a];x;a]]; \]
\[ \text{eq}[\text{car}[fn]; \text{LAMBDA}] \rightarrow \text{eval}[\text{caddr}[fn]; \text{pairlis}[\text{cadr}[fn]; x; a]]; \]
\[ \text{eq}[\text{car}[fn]; \text{LABEL}] \rightarrow \text{apply}[\text{caddr}[fn]; x; \text{cons}[\text{cons}[\text{cadr}[fn]; \text{caddr}[fn]]; a]]] \]

\[ \text{eval}[e; a] = [\text{atom}[e] \rightarrow \text{cdr}[\text{assoc}[e; a]]; \]
\[ \text{atom}[\text{car}[e]] \rightarrow \]
\[ [\text{eq}[\text{car}[e]; \text{QUOTE}] \rightarrow \text{cadr}[e]; \]
\[ \text{eq}[\text{car}[e]; \text{COND}] \rightarrow \text{evcon}[\text{cdr}[e]; a]; \]
\[ T \rightarrow \text{apply}[\text{car}[e]; \text{evlis}[\text{cdr}[e]; a]; a]]; \]
\[ T \rightarrow \text{apply}[\text{car}[e]; \text{evlis}[\text{cdr}[e]; a]; a]] \]
Eval/Apply in Logo

Eval

- eval_line
- logo_eval

Call expressions

Apply

- apply_procedure
- collect_args
- logo_apply

User-defined procedures

Operand expressions

Monday, November 7, 2011