Review

Game playing

• Difference from simple state space
  – Agent does not have sole control of environment

• Result is min/max tree
  – Nodes are game states, arcs are moves
  – Each player controls only some levels
  – Leaves are end-states of game, with values
Review

• If game is small enough, back up leaf values with mini-max
• Otherwise, stop before leaves and use a static evaluation function
• Faster search: alpha-beta

LISP

• Invented by McCarthy in 1958.
• Much work on programming languages and interactive programming environments was done in Lisp
  – E.g. functional and object-oriented programming
  – Java is largely influenced by Lisp
Invented in Lisp

• Conditionals
• Function as a data type
• Recursion
• Variable as pointer to typed data
• Garbage collection
• Program = expression
• Symbol data type
  – Uniquified strings
• Code = tree of symbols

Current dialects

• Scheme: small and elegant
• Common Lisp: large and powerful
• Scripting languages
  – Emacs
  – Gimp
  – …
Why LISP for AI?

- Good for "symbol manipulation".
  - Symbols, lists, trees, binary trees built in.
  - Garbage collector built in.
- Good for exploratory programming.
  - Interactive and incremental
  - Can print and read most data types
  - Extremely extensible.
- Code is data
  - Code can easily create new code
  - A data structure can contain code

Code = Data

- Allows programming technique “data driven programming”
  - Many different ways to do a task, e.g. differentiate
  - Pieces of code in a data structure
  - Use data as a key to look up code in data structure
Extremely Extensible

• A historical approach to AI programming: to solve problem X:
  – Implement a language to solve problems of same general class as X
  – Write a program for X in this language

• Modern version: extend lisp to be this language

Review of Basic Ideas of Lisp

The basic data structure is a list, written with parens and no commas:

(1 4 3)

Any lisp data type can be an element of any list

(1 AB 3.14 "abc" (1 4 3))
Access functions:

– car or first: first element of the list
  – car of (1 2 3) is 1
  – car of ((1 2) 3 (4)) is (1 2)

– cdr or rest: all of list except the first
  – cdr of (1 2 3) is (2 3)
  – cdr of ((1 2) 3 (4)) is (3 (4))

– cons: create a list from a car and a cdr
  – cons of (1 2) and (3 (4)) is ((1 2) 3 (4))
  – The empty list is nil. cons of 1 and nil is (1)

A program is simply one or more expressions to evaluate.

• (+ 4 6) value is 10
Expressions

• An expression is simply a list.
  – Car is function to call
  – Cdr is list of arguments
    (+ 3 4)
  – numbers evaluate to themselves, symbols are variables
    (+ 3 x)

Normally, to evaluate a list,

• Car specifies a function: figure out which
• Evaluate each element of the cdr
• Pass those values as the arguments to the functions specified by the car
  (+ (+ 3 4) (* 5 6))
  + means <code for add>
  Evaluate (+ 3 4) -> 7
  Evaluate (* 5 6) -> 30
  Pass 7 and 30 as args to <code for add>
Special Forms

• Some things that look like function names are "special" and do not follow these evaluation rules

• quote inhibits evaluation
  (list 1 (+ 2 3)) -> (1 5)
  (list 1 '(+ 2 3)) -> (1 (+ 2 3))

• if is a conditional
  (if (not (= x 0))
    (/ y x))
  'impossible)

Common Lisp vs Scheme

• In scheme: car evaluated to function with normal eval

• In common lisp: 2 Kinds of “Evaluation”
  – In cdr: normal eval-as-data
  – In car: get meaning-as-function
  – If you want meaning-as-function in cdr
    – Use (function 'foo)
    – Abbreviated #'foo
  – If you want effect of normal eval in car, use funcall
Data types in cl

- **Arrays**
  
  ```lisp
  (setf a (make-array '(3 4)))
  (aref a 0 2)
  (setf (aref 0 2) 3)
  ```

- **Structures**
  
  ```lisp
  (defstruct rectangle x y height width)
  (let ((rect (make-rectangle :x 20
          :width 10 :height 4))))
    (setf (rectangle-height rect) 2)
    (rectangle-x rect))
  ```

- **Others:** numbers of many kinds, characters, hash tables, ...

---

Booleans:

- in common lisp, nil is false, anything else is true
  
  ```lisp
  (or x 0) = (if x x o)
  (and (numberp x) (+ x 1)) = (if (numberp x)
        (+ x 1)
      nil)
  ```
Example: hexapawn

• Hexapawn:
  – 3x3 chess board, each player has 3 pawns
  – pawns move as in chess
  – win: get a pawn to other side of board or opponent's turn and opponent has no legal move.