Tools for Novice AI Programmers

Simon Lynch
University of Teesside
background issues

• aim: level 3+ AI with practical content
• students
  ◦ less computer science
  ◦ more games, web, etc
• their interests
  ◦ games characters & behaviours
  ◦ user modelling, MMD, agents, etc
student background

• computer science students
  ○ functional programming
  ○ formal methods
  ▼ some symbolics, sets, tuples, etc

• other disciplines
  ○ C++ (games students)
  ○ Java or VB (others)
identified problems

- practical symbolic AI programming
- transition to symbolics
- barriers greater for non-CS students

but...

symbolic and/or declarative representations not the problem
symbolic example

• numeric function $f$
  
  $f(73.6) = 220.8$
  
  $f(14.79) = 44.37$

• symbolic function $g$
  
  $g((\text{the cat}) \ (\text{the dog}) \ (\text{the frog}))$
  
  $= (\text{cat} \ \text{dog} \ \text{frog})$
why the problem?

• background in mathematics?

• reinforced by introductory programming?

• utility of AI functions only understood in wider context (eg: conflict resolution)?
difficulties

• solving AI problems at a conceptual level

• specifying problems / solutions (in any form)

• using the target programming language... (Lisp)
difficulties

using the target programming language
• adapting to new language features (eg: CLOS & mapping)
• deconstructing symbolic data-structures (eg: pulling apart nested associations)
• unfamiliar control structures (notably recursion)
• using prior knowledge to help with Lisp
strategies

• limit scope of language

• developing marketable programming expertise a secondary goal to gaining practical experience in AI

• provide tools for data manipulation deconstruction, etc, etc
tools – set operators

<table>
<thead>
<tr>
<th>set-difference</th>
<th>$-$</th>
</tr>
</thead>
<tbody>
<tr>
<td>union</td>
<td>$+$</td>
</tr>
<tr>
<td>intersection</td>
<td>$*$</td>
</tr>
<tr>
<td>subsetp</td>
<td>$&lt;=$</td>
</tr>
</tbody>
</table>

> (union '((a cat)(a dog)(a frog)) '((a frog)(a bat)(a dog)))
→((a frog) (a dog) (a cat) (a frog) (a bat) (a dog))

> ($+ '((a cat)(a dog)(a frog)) '((a frog)(a bat)(a dog)))
→((a frog) (a dog) (a cat) (a bat))
mapping

(defvar fruit
  '(((color cherry red)
      (color apple green)
      (color banana yellow)
      (color kiwi green)
    ))
  > (mapcar #'second fruit)
  → (cherry apple banana kiwi)
mapping example

(defun intersect1 (s1 s2)
  (cond ((null s1) nil)
        ((member (first s1) s2)
         (cons (first s1) (intersect1 (rest s1) s2)))
        (t (intersect1 (rest s1) s2)))
)

(defun intersect2 (s1 s2)
  (remove-if-not
    #'(lambda (x)
        (member x s2)) s1))
mapping alternative

remove-if (filter data removing x when cond)

remove-if-not (filter data removing x unless cond)

mapcar with lambda form

(mapcar with lambda form)

(mapcar with lambda form)

reduce (filter data collecting x as exp)

reduce (filter data reducing x with fn)
mapping example

(defun intersect2 (s1 s2)
  (remove-if-not
    #'(lambda (x)
        (member x s2)) s1))

(defun intersect3 (s1 s2)
  (filter s1 removing x
        unless (member x s2)))
(defvar country
  '(((africa (botswana (capital . gaborone)
                (population . 2m))
     (zimbabwe (capital . harare)
                (population . 11m)))))

(-> country 'africa 'zimbabwe 'capital)
  -> harare
matching

(mlet ( `(the ?subj ?verb ??obj)
       `(the cat chased the rat)
     )
  (match>>(`(??obj was ?verb by a ?subj)))
  → (the rat was chased by a cat)
match methods

(defmatch calculate ((?x plus ?y))
  (+ #?x #?y))

(defmatch calculate ((?x minus ?y))
  (- #?x #?y))

> (calculate '(5 plus 3)) → 8
> (calculate '(5 minus 3)) → 2
match iterators

(defvar data
 '(((isa b1 box)  (color b1 red)  (size b1 large))
 (isa b2 box)  (color b2 red)  (size b2 small))
 (isa b3 box)  (color b3 blue)  (size b3 small))
 (isa b4 box)  (color b3 blue)  (size b4 small)))

> (foreach '(size ?x small) data)
    (format t "~a is small, "  #?x)
    #?x)

→ b2 is small, b3 is small, b4 is small
→ (b2 b3 b4)
match iterators

(defvar data
  '((isa b1 box)  (color b1 red)   (size b1 large)
     (isa b2 box)  (color b2 red)   (size b2 small)
     (isa b3 box)  (color b3 blue)  (size b3 small)
     (isa b4 box)  (color b3 blue)  (size b4 small) ))

> (forevery '(((isa ?b box) (color ?b red)) data)
      (format t "~a is a red block, " #?b)
      (list 'red-block #?b))

→ b1 is a red block, b2 is a red block
→ ((red-block b1) (red-block b2))
rules example

expert system rules – a typical unhappy compromise...

(rule 32 (has fido hair) => (is fido mammal))

vs.

(rule 32 (has ?x hair) => (is ?x mammal))

(rule 15 (parent ?a ?b) (parent ?b ?c)
 => (grandparent ?a ?c))
rules example

(setf family
    '(((parent sarah tom) (parent steve joe)
        (parent sally sam) (parent ellen sarah)
        (parent emma bill) (parent rob sally)))

    family)
    (match>> '(grandparent ?a ?c)))

→ ((grandparent ellen tom)
    (grandparent rob sam))
rules example

(defun apply-rule (r facts)
  (mlet ('(rule ?n ??antecedents => ??consequents) r)
    (forevery (#?antecedents facts)
      (setf facts ($+ (match>> #?consequents) facts)))
    facts))

> (apply-rule '(rule 15 (parent ?a ?b) (parent ?b ?c)
    => (grandparent ?a ?c))
  family)

→ ((grandparent rob sam) (grandparent ellen tom) ....)
operators example

state changing operators – another unhappy compromise?

(pickup-cup (at table) (hand empty) (on cup saucer)
    => (at table) (holding cup) (top saucer clear))

vs.

(defvar pick-up ; pick up object ?x
    '((pre (holds nil) (cleartop ?x) (supports ?y ?x))
        (del (holds nil) (supports ?y ?x))
        (add (holds ?x) (cleartop ?y))
    ))
operators example

(defun apply-op (op object world)
  (let ((pre (-> op 'pre))
        (del (-> op 'del))
        (add (-> op 'add)))
    (all-present (pre world `((x ,object)))
                ($+ (match>> add)
                    ($- world (match>> del)))))

use...
> (defvar blocks '((isa b1 block) (cleartop p1) (holds nil)
                      (isa p1 pyramid) (supports b1 p1) .... ))
> (apply-op pick-up 'p1 blocks)
→ ((cleartop b1) (holds p1) (cleartop p1)
    (isa p1 pyramid) (isa b1 block) ....))
results & feedback

monitored over 3-4 years...
• feedback positive
• attendance improved
• participation improved
• greater take up as module option
• CS / non-CS division removed
• improved assessment results...
assessment

formative...
3 tasks of 4-5 weeks, 10-15 parts each
completion rates improved from (approx) 30% completion to 70%

summative...
theoretical 3 hour exam
results improved from (approx) 50% average to 60%
Lisp tools available at

www.agent-domain.org