MIT 6.001 Fall 1997
A. R. Meyer October 9, 1997

Pattern Matching: Examples

```
EXAMPLE 1:          THE ELIZA (DOCTOR) PROGRAM

(doctor)                  ; START the session

(hello, what can i do for you?) ; ELIZA

(i need help)                   ; USER
(why do you say you need help ?)  ; ELIZA

(because my mother is driving me crazy)
(tell me more about your mother)

(she makes me feel guilty)
(is it usual for you to feel guilty ?)

(not really)
(what makes you say not really)
```

```
(because I am ok when she is not around)
(are you feeling defensive?)

(maybe)
(why are you being short with me?)

(will you stop asking me so many questions)
(would you ask that of anyone else besides me?)

quit
; Value: "Your session is over.  Good luck."
```

```
(define eliza-rules
  (list
   (make-SIMPLE-rule
    '(~ mother ~)
    '(tell me more about your mother))

   (make-simple-rule
    '(~ because ~)
    '(are you feeling defensive?))

   (make-simple-rule
    '(~ you ~)
    '(would you ask that of anyone else besides me?))

   (make-simple-rule
    '(?)
    '(Why are you being short with me?))

   (make-simple-rule
    '(~ feel ?adverb ~)
    '(Is it usual for you to feel ?adverb ?))
```

Rules for Matching

ATOMs MUST MATCH EXACTLY

```
(define (is-atom? p)
  (or (number? p)
      (boolean? p)
      (and (symbol? p)
           (not (qmark-variable? p))
           (not (tilde-variable? p)))))
```

?VAR: MATCHES ANY SINGLE DATUM

```
PATTERN:         (?F   * (  ?G    + ?H))

DATUM:        ((X + 1) * ((A * B) +  C))

MATCH DETERMINES A
               DICTIONARY: ((?F (X + 1))
                            (?G (A * B))
                            (?H C))

SIMPLE RULE:    (PATTERN TEMPLATE)

TEMPLATE:   ((   ?F   *   ?G)    + (   ?F   * ?H))
```

APPLY THE RULE MEANS "INSTANTIATE" TEMPLATE USING DICTIONARY FROM THE MATCH TO GET:

```
            (((X + 1) * (A * B)) + ((X + 1) *  C))
```

~VAR: MATCHES ANY NUMBER OF DATA ITEMS

```
PATTERN:                    (~A ~B ?C ~D)
DATUM:                      (1 2 3 4)

LIST OF ALL 10 DICTIONARIES FROM MATCHES:

(((~A ()) (~B ()) (?C 1) (~D (2 3 4)))
 ((~A ()) (~B (1)) (?C 2) (~D (3 4)))
 ((~A ()) (~B (1 2)) (?C 3) (~D (4)))
 ((~A ()) (~B (1 2 3)) (?C 4) (~D ()))
 ((~A (1)) (~B ()) (?C 2) (~D (3 4)))
 ((~A (1)) (~B (2)) (?C 3) (~D (4)))
 ((~A (1)) (~B (2 3)) (?C 4) (~D ()))
 ((~A (1 2)) (~B ()) (?C 3) (~D (4)))
 ((~A (1 2)) (~B (3)) (?C 4) (~D ()))
 ((~A (1 2 3)) (~B ()) (?C 4) (~D ())))
```

```
TEMPLATE:         (~A - ~B - ?C - ~D)
INSTANTIATIONS
      (- - 1 - 2 3 4)
      (- 1 - 2 - 3 4)
      (- 1 2 - 3 - 4)
      (- 1 2 3 - 4 -)
      (1 - - 2 - 3 4)
      (1 - 2 - 3 - 4)
      (1 - 2 3 - 4 -)
      (1 2 - - 3 - 4)
      (1 2 - 3 - 4 -)
      (1 2 3 - - 4 -)
```

PURE TILDE MATCHES ANY NUMBER OF THINGS, BUT DOES NOT CONTRIBUTE TO THE DICTIONARY

```
PATTERN:      '(~ mother ?what ~)

DATUM:        (mother this and mother that)

DICTIONARIES: ((?what this)
               (?what that))
```

Eliza Rules (Cont'd)

```
(make-simple-rule
 '(I ~phrase)
 '(why do you say you ~phrase ?))

(make-RESTRICTED-rule
 '(~ to ?verb ~)
 '(When did you start to ?verb ?)
 CHECK-FOR-VERB) ;RESTRICTION-PRED: DICT --> BOOL
))

(define (CHECK-FOR-VERB dict)
      (memq (lookup '?verb dict)
            '(dance run sing cry laugh)))
```

Example 2: SYMBOLIC DERIVATIVES --- D(f) = df/dx

```
(define deriv-rules

                ;D(F + G) = D(F) + D(G)
  (make-simple-rule
   '(deriv (?F + ?G) ?var)
   '((deriv ?F ?var) + (deriv ?G ?var)))

                ;D(F * G) = (F * D(G)) + (G * D(F))
  (make-simple-rule
   '(deriv (?F * ?G) ?var)
   '((?F * (deriv ?G ?var))
     +
     (?G * (deriv ?F ?var))))

                      ;D(n) = 0
  (make-RESTRICTED-rule
   '(deriv ?num ?var)
   0
   NUMBER-RESTRICTION)
        .
        .
        .

(define (NUMBER-RESTRICTION dict)
 (number? (lookup ?num dict)))
```

```
                          ;D(y) = 1 when y = x
(make-restricted-rule
 '(deriv ?Y ?X)
 1
 (lambda (dict)
   (eq? (lookup '?Y dict)
        (lookup '?X dict))))
                          ;D(y) = 0 when y =\= x
(make-restricted-rule
 '(deriv ?Y ?X)
 0
 (lambda (dict)
   (let ((datum (lookup '?Y dict)))
     (and (symbol? datum)
          (not (eq? datum (lookup '?X dict)))))))
```

```
Example 3:     ;;ARITHMETIC SIMPLIFICATION

(define simplify-rules
  (make-simple-rule '(0 + ?F) '?F)

  (make-simple-rule '(1 * ?G) '?G)
  (make-simple-rule
   '(?H * (?F + ?G))
   '((?H * ?F) + (?H * ?G)))

  (make-restricted-rule
   '(?E + ?F) '(2 * ?E)
   (lambda (dict)           ; E = F
     (equal? (lookup '?E dict)
             (lookup '?F dict))))

  (make-RESTRICTED-rule    ;+NUMS-LEFT-RULE
   '(?F + ?N) '(?N + ?F)
   N-NUM/F-NOT-NUM-RESTRICTION)

  rule:prim-+ ...)
```

```
RULE-PRIM-+:
   (make-simple-rule '(0 + 0) 0)
   (make-simple-rule '(0 + 1) 1)
   (make-simple-rule '(1 + 0) 1)
   (make-simple-rule '(1 + 1) 2)
   (make-simple-rule '(0 + 2) 2)
   (make-simple-rule '(2 + 0) 2)
   (make-simple-rule '(1 + 2) 3) ...
```

```
(define rule:prim-+
  (make-GENERAL-rule
   '(?N + ?M)
   SUM-N-M))        ;RESULT PROCEDURE: DICT --> DATUM

(define (SUM-N-M dict)
     (let ((N (lookup 'N dict))
           (M (lookup 'M dict)))
       (if (and (number? N)
                (number? M))
           (+ N M)
           (fail))))
```

GENERAL RULES: ESSENTIAL, BUT MAY UNDERMINE CLEAN PATTERN-MATCHING STYLE

Example 4: RULES FOR DESUGARING QUOTED LISTS

REMEMBER '(...) abbreviates (QUOTE (...))

```
(do-rules quote-rules '(quote ((1 a) ((b) c))))
Value: (list (list 1 'a) (list (list 'b) 'c))
```

```
(define quote-rules

        ; '(E F_1 F_2 ...)  ==> (CONS 'E '(F_1 F_2 ...))
  (make-simple-rule
   '(quote (?E ~FF))
   '(cons (quote ?E) (quote (~FF))))

        ; '3  ==>  3       [3 is "SELF-EVALUATING"]
  (make-restricted-rule
   '(quote ?S) '?S
  (lambda (dict)
    (let ((S (lookup '?S dict)))
      (or (number? S)
          (boolean? S)))))

          ;;CONS Simplification:
             ; (CONS E '()) ==> (LIST E)
  (make-simple-rule
   '(cons ?E '()) '(list ?E))

  (make-simple-rule
   '(cons ?E (list ~FF)) '(list ?E ~FF))

  (make-simple-rule '(list) '()))
```

EXAMPLE 5: EXTENSION OF SCHEME DEFINITION FORMAT

```
(define (append L1 L2)
  (if (null? L1)
      L2
      (cons (car L1)
            (append (cdr L1) L2))))

(define-match
  ((append () ?L2)
      ?L2)
  ((append (?x ~L1) ?L2)
      (?x (append ~L1 ?L2))))
```

```
(define (append L . LL)
  (cond ((null? LL) L)
        ((null? L) (apply append LL))
        (else
         (cons (car L)
               (apply append
                      (cons (cdr L) LL))))))

(define-match
  ((append ?L) ?L)
  ((append () ~LL) (append ~LL))
  ((append (?x ~L) ~LL) (?x (append (~L) ~LL))))
```

Pattern Matching: Implementation

```
(define (match pat dat)
  (cond
   ((qmark? pat)
    (list the-empty-dictionary))
   ((qmark-variable? pat)
    (list (make-singleton-dictionary pat dat)))
   ((or (is-atom? pat) (null? pat))
    (if (equal? pat dat)
        (list the-empty-dictionary)
        '()))           ;FAIL
   ((is-atom? dat) '()) ;FAIL -- pat is not an atom
   ((null? dat) (match-null pat))
   (else (match-pairs pat dat))))  ;PAT & DAT are pairs
```

```
(define (match-pairs pat dat)   ;PAT & DAT are pairs
  (let ((pat0 (car pat)))
    (cond
     ((tilde? pat0)
      (match-initial-tilde (cdr pat) dat))
     ((tilde-variable? pat0)
      (match-initial-tildevar pat0 (cdr pat) dat))
     (else
      (let ((dicts1 (match pat0 (car dat))))
        (if (null? dicts1)
            '()
            (all-pairs dicts1
                       (match (cdr pat) (cdr dat))
                       append))))))))
```

```
(all-pairs '(x1 ... xn) '(y1 ... ym) op) =
     (list
       (op x1 y1) (op x1 y2) ... (op x1 ym)   --|
                           .                    |
                           .                    |  ONE BIG LIST
                           .                    |
       (op xn y1) (op xn y2) ... (op xn ym))  --|
```

```
To match (~ PAT1 PAT2 ..) against (DAT1 DAT2 ...)
case 1: ~ matches nothing and
           (PAT1 PAT2 ...) matches (DAT1 DAT2 ...)
case 2: ~ matches against one or more of DAT1 DAT2 ....
          This happens exactly when (~ PAT1 PAT2 ...)
          matches (DAT2 ...).
```

PURE TILDE

```
(define (match-initial-tilde pat dat)
  (define (aux dat)   ;matches (~ PAT1 PAT2 ...) against DAT
    (append
     (match pat dat)  ;~ not used to match anything
     (if (null? dat)
         '()         ;already have dicts from null DAT
                     ;Now check match when ~ is used to match
                     ;a sequence starting with DAT1
         (aux (cdr dat))))))
```

NAMED TILDE VARIABLE

```
To match (~var PAT1 PAT2 ..) against (DAT1 DAT2 ...)
case 1: ~var matches nothing and
         (PAT1 PAT2 ...) matches (DAT1 DAT2 ...)
case 2: ~var matches against one or more of DAT1 DAT2 ....
         To see how,
          (i) match (~var PAT1 PAT2 ...) against
               (DAT2 ...) and then
          (ii) put DAT1 at the start of the sequence
                that ~var matches in each dictionary from (i)
```

```
(define (match-initial-tildevar tilde-var pat dat)
  (define (aux dat)      ;matches (~VAR PAT) against DAT
    (append
                         ;dicts with ~VAR matching empty
     (map (dictionary-reviser tilde-var '())
          (match pat dat))
     (if (null? dat)
         '()               ;already have dicts from null DAT
         (let ((dat1 (car dat)))  ;~VAR matches (DAT1 ...)
           (map
            (lambda (dict)
              ((dictionary-reviser
                tilde-var
                (cons dat1 (lookup tilde-var dict)))
               dict)
              (aux (cdr dat))))))))
           ;dicts when ~VAR matches the
           ;beginning items of (cdr DAT):
  (aux dat))
```

```
                        ;;APPLYING RULES

(define (instantiate temp dict)
  (cond ((or (qmark-variable? temp)
             (tilde-variable? temp))
         (lookup temp dict))
        ((or (is-atom? temp) (null? temp))
         temp)
        (else
         (let ((temp1 (car temp)))
           ((if (tilde-variable? temp1)
                append
                cons)
            (instantiate temp1 dict)
            (instantiate (cdr temp) dict))))))
```

```
;Do SOME APPLICATION of the rule to the ENTIRE DATUM

   ;(We choose the first dictionary of the match.)
(define (SOME-RULE-APPLICATION rule datum)
  (let ((dicts (match (rule.pattern rule) datum)))
    (if (null? dicts)
        (fail)
        (cond
         ((simple? rule)
          (instantiate
           (rule.template rule) (car dicts)))
```

```
((restricted? rule)
 (let ((dict-ok? (rule.restriction rule)))
   (define (search maybe-ok-dicts)
     (if (null? maybe-ok-dicts)
         (fail)
         (let ((dict (car maybe-ok-dicts)))
           (if (dict-ok? dict)
               (instantiate
                (rule.template rule)
                dict)
               (search (cdr maybe-ok-dicts))))))
   (search dicts)))
((general? rule)...)
 (else (error "unknown rule" rule))))))
```

```
  ;Do some application of some rule to SOME SUB-DATUM

  ;(We choose the first rule that has a match.)
(define (SOME-INNER-APPLICATION rules datum)
  ...)

             ;keep applying rules UNTIL
             ;NO RULE APPLIES ANYWHERE

(define (SOME-FINAL-FORM rules datum)
  (let ((result (some-inner-application rules datum)))
    (if (fail? result)
        datum
        (some-final-form rules result))))

(define DO-RULES some-final-form)
```

Closing Sound Bite: Pattern Matching is a Fundamental Technique in Computing