specifying a grammar in Lisp with the matcher

NB: many of the examples here use the matcher and utility functions available from www.agent-domain.org

grammars

grammars can be used to generate 'sentences' (see L-systems examples on the NetLogo pages of agent-domain) or to produce phrase-structures (parse trees) from strings of words. This example investigates the latter, using the matcher to define the grammar rules.

the problem...

given a sentence like "*a cat chased the large rat*" we want to produce a phrase structure tree something like...

```
> (sentence '(a cat chased the large rat))
(sentence (noun-phrase (det a) (noun cat))
          (verb-phrase (verb chased)
                       (noun-phrase
                            (det the)
                            (adj large)
                            (noun rat))))
```

simple categories

We can deal with individual words fairly easily, for example if "cat" is a noun then parsing the word "cat" on its own should return...

```
(cat noun)
```

One way of doing this is to have a list of all nouns & use a category checking function...

```
(defparameter *nouns* '(cat bat rat kipper melon))

(defun check-category (cat-name word word-set)
  (if (member word word-set)
      (list cat-name word)
    nil
    ))

(defun noun (x) (check-category 'noun x *nouns*))

> (noun 'cat)
(noun cat)

> (noun 'thirteen)    ;; returns nil if it fails
nil
```

We can do something similar for other categories of word...

```
;; word definitions

(defparameter *nouns* '(cat bat rat kipper melon))
(defparameter *verbs* '(ate chased))
(defparameter *dets*  '(the a every))
(defparameter *adjs*  '(large small))

;; word checks

(defun noun (x) (check-category 'noun x *nouns*))
(defun verb (x) (check-category 'verb x *verbs*))
(defun det (x) (check-category 'det x *dets*))
(defun adj (x) (check-category 'adj x *adjs*))

> (noun 'thirteen)
nil

> (det 'every)
(det every)

> (adj 'every)
nil

> (adj 'small)
(adj small)
```

non-terminal categories

We need a different style of solution for non-terminal categories like noun-phrases (which are made up of multiple words). The matcher allows predicates to be used to enforce restrictions on match patterns...

```
> (mlet ( '(??a ?x ??b) '(a b c 10 d e f) )
        (match>> '((a ?a)(b ?b)(x ?x))))
((a nil) (b (b c 10 d e f)) (x a))

> (mlet ( '(??a ?(x numberp) ??b) '(a b c 10 d e f) )
        (match>> '((a ?a)(b ?b)(x ?x))))
((a (a b c)) (b (d e f)) (x 10))
```

If we use some variation of a predicate which returns something other than t to indicate success, the matcher binds this non-nil value to the relevant match variable. For example...

```
(defun nsquarep (x)
    (and (numberp x)
            (* x x)))

> (nsquarep 'banana)
nil
```

```
> (nsquarep '10)
100

> (mlet ( '(??a ?(x nsquarep) ??b) '(a b c 10 d e f) )
        (match>> '((a ?a)(b ?b)(x ?x))))
((a (a b c)) (b (d e f)) (x 100))
```

We can use this to build parse expressions for our non-terminal categories.

```
cg-user(50): (mlet ( '(?(d det) ?(n noun)) '(the cat) )
                   (match>> '((d ?d)(n ?n))))
((d (det the)) (n (noun cat)))

(defmatch noun-phrase ((?(d det) ?(n noun)))
  (match>> '(noun-phrase ?d ?n)))

cg-user(51): (noun-phrase '(the cat))
(noun-phrase (det the) (noun cat))

cg-user(52): (noun-phrase '(kipper melon))
nil
```

There is no problem with defining multiple noun-phrase rules because it is a defmatch form...

```
(defmatch noun-phrase ((?(d det) ?(a adj) ?(n noun)))
  (match>> '(noun-phrase ?d ?a ?n)))

cg-user(54): (noun-phrase '(a large cat))
(noun-phrase (det a) (adj large) (noun cat))

cg-user(55): (noun-phrase '(every small kipper))
(noun-phrase (det every) (adj small) (noun kipper))

cg-user(56): (noun-phrase '(the melon))
(noun-phrase (det the) (noun melon))
```

Now we define another couple of forms to build on noun-phrase...

```
;; sentence rule
(defmatch sentence ((??(np noun-phrase) ??(vp verb-phrase)))
  (match>> '(sentence ?np ?vp)))

;; verb phrase
(defmatch verb-phrase ((?(v verb) ??(np noun-phrase)))
  (match>> '(verb-phrase ?v ?np)))
```

```
cg-user(57): (sentence '(the large cat ate a small kipper))
(sentence (noun-phrase (det the) (adj large) (noun cat))
          (verb-phrase (verb ate)
                       (noun-phrase
                          (det a) (adj small) (noun kipper))))
```

the finished work...

```
;; sentence rule
(defmatch sentence ((??(np noun-phrase) ??(vp verb-phrase)))
  (match>> '(sentence ?np ?vp)))


;; noun-phrase rules
(defmatch noun-phrase ((?(d det) ?(n noun)))
  (match>> '(noun-phrase ?d ?n)))

(defmatch noun-phrase ((?(d det) ?(a adj) ?(n noun)))
  (match>> '(noun-phrase ?d ?a ?n)))


;; verb phrase
(defmatch verb-phrase ((?(v verb) ??(np noun-phrase)))
  (match>> '(verb-phrase ?v ?np)))


;; word definitions
(defparameter *nouns* '(cat bat rat kipper melon))
(defparameter *verbs* '(ate chased))
(defparameter *dets*  '(the a every))
(defparameter *adjs*  '(large small))


;; word checks
(defun noun (x) (check-category 'noun x *nouns*))
(defun verb (x) (check-category 'verb x *verbs*))
(defun det (x) (check-category 'det x *dets*))
(defun adj (x) (check-category 'adj x *adjs*))


;; category helper
(defun check-category (cat-name word word-set)
  (if (member word word-set)
      (list cat-name word)
    nil
    ))
```