

Variables: symbols beginning with the characters u,v,w,x,y, or z.

E.g., 'v3 'uabc 'x

Constants: symbols other than variables

E.g., 'c1 'k 'fx 't17

Patterns:

```
(define p1 '(a b (q y) (a z) w) )
(define p2 '(v b x (w z) w) )
(define p3 '(v b c (a c) w) )
(define p4 '(x b c (x v) w) )
(define p5 '((a w) b c (v x) v) )
(define p6 '(x (x x x) v (v v v) ) )
(define p7 '((y y y) z (z z z) vz) )
```

```
(define variable (lambda (x)
  (and (symbol? x)
       (string>=? (symbol->string x) "u") ) ))
```

```
(define constant (lambda (c)
  (and (symbol? c)
       (string<? (symbol->string c) "u") ) ))
```

```
(define pattern (lambda (e) (pair? e)))
```

The *difference* of two expressions:

```
(define diff (lambda (a b)
  (cond ((eq? a b) ())
        ((variable b) (list a b))
        ((constant b) (list b a))
        ; now, b must be a pattern
        ((not (pair? a)) (list b a))
        ( (diff (car a) (car b))) ; if non-null, return it. Else,
          (#t (diff (cdr a) (cdr b))) ) ) ) ; return diff of cdr's.
```

A difference is *reducible* if
the 2-nd element is a variable **not** occurring in the first

```
(define reducible? (lambda (diffpair)
  (and (not (null? diffpair))
        (variable (cadr diffpair))
        (not (occur (cadr diffpair) (car diffpair))) ) ) )

(define occur (lambda (x y) ; x must be a variable
  (cond ((eq? x y) #t)
        ((not (pair? y)) ())
        (#t (or (occur x (car y)) (occur x (cdr y)))) ) ) )
```

UNIFYING expressions: can we unify, or match p1 and p2 ?

p1 = '(a b (q y) (a z) w))
p2 = '(v b x (w z) w))

YES, if:

v becomes a and
x becomes (q y) and
w becomes a

The substitution of expressions for variables is represented as a list of *components*. Each component is a two-element list in which the first element replaces the second (required to be a variable).

The substitution above would be represented as:

((a v) ((q y) x) (a w))

UNIFICATION (informal algorithm)

To unify expressions A and B, we start with

A, B, (diff A B), and the empty substitution ().

While (diff A B) is reducible, we:

modify A and B as indicated by (diff A B)

augment substitution S as indicated by (diff A B)

compute a new (diff A B)

EndWhile

EXAMPLE :

A = (P X (F (G Y)) (F X))

B = (P (H Y Z) (F Z) (F (H U V)))

S = ()

(diff A B) = ((H Y Z) X)

A = (P (H Y Z) (F (G Y)) (F (H Y Z)))

B = (P (H Y Z) (F Z) (F (H U V)))

S = (((H Y Z) X))

(diff A B) = ((G Y) Z)

$$A = (P \ (H \ Y \ (G \ Y)) \ (F \ (G \ Y)) \ (F \ (H \ Y \ (G \ Y))))$$

$$B = (P \ (H \ Y \ (G \ Y)) \ (F \ (G \ Y)) \ (F \ (H \ U \ V \ \ \ \)))$$

$$S = (((H \ Y \ (G \ Y)) \ X) \ ((G \ Y) \ Z))$$

$$(\text{diff } A \ B) = (Y \ U)$$

$$A = (P \ (H \ Y \ (G \ Y)) \ (F \ (G \ Y)) \ (F \ (H \ Y \ (G \ Y))))$$

$$B = (P \ (H \ Y \ (G \ Y)) \ (F \ (G \ Y)) \ (F \ (H \ Y \ V \ \ \ \)))$$

$$S = (((H \ Y \ (G \ Y)) \ X) \ ((G \ Y) \ Z) \ (Y \ U))$$

$$(\text{diff } A \ B) = ((G \ Y) \ V)$$

$$A = (P \ (H \ Y \ (G \ Y)) \ (F \ (G \ Y)) \ (F \ (H \ Y \ (G \ Y))))$$

$$B = (P \ (H \ Y \ (G \ Y)) \ (F \ (G \ Y)) \ (F \ (H \ Y \ (G \ Y))))$$

$$S = (((H \ Y \ (G \ Y)) \ X) \ ((G \ Y) \ Z) \ (Y \ U) \ ((G \ Y) \ V))$$

$$(\text{diff } A \ B) = ()$$

```

(define unify (lambda (a b)
  (cond
    ((and (pattern a) (pattern b)
          (not (eq? (length a) (length b)))) )
; both patterns, and unequal length, so fail.
    ('not-unifiable)
    (#t (let ((diffab (diff a b))
              (onlydiff? (or (not (pattern a))
                              (not (pattern b)) ) ) )
          (cond ((null? diffab) ()) ; No diff - done!
                ((not (reducible? diffab)) 'not-unifiable)
                (onlydiff? (list diffab)))
; one of a,b isn't a pattern - this is the only difference,
; otherwise, both are patterns. Call unify on
; [diffab applied to a and to b] and return the
; composition of diffab and the result.
          (#t (compose
                (unify (subst (car diffab)
                              (cadr diffab)
                              a )
                        (subst (car diffab)
                              (cadr diffab)
                              b ) )
                diffab )) ) ) ) ) ) )

```

; the variable of diffpair must not occur in sub,
; because diffpair is *always* added as a component.

```
(define compose (lambda (sub diffpair)
  (cond ((eq? sub 'not-unifiable) 'not-unifiable)
        ((null? sub) (list diffpair))
        (#t (cons (list (subst (car diffpair)
                               (cadr diffpair)
                               (caar sub) )
                    (cadar sub) )
                  (compose (cdr sub)
                          diffpair ) )) ) ) )
```

```
(define subst (lambda (x y z)
  (cond ((eq? y z) x) ; y is always a variable
        ((not (pair? z)) z) ; y doesn't occur in z
        (#t (cons (subst x y (car z))
                  (subst x y (cdr z)) ) ) ) )
```

```
(define appsub (lambda (sub e)
  (cond ((null? sub) e)
        (#t (appsub (cdr sub)
                    (subst (caar sub)
                          (cadar sub)
                          e ) ) ) ) )
```

```

p1 = '(a      b      (q y)      (a z)      w) )
p2 = '(v      b      x      (w z)      w) )
p3 = '(v      b      c      (a c)      w) )
p4 = '(x      b      c      (x v)      w) )
p5 = '((a w)  b      c      (v x)      v) )
p6 = '(x      (x x x) v      (v v v)      ) )
p7 = '((y y y) z      (z z z)  vZ      ) )

```

```
1 ]=> (map constant (list 'c1 'c2 'x 'y p5 p7))
```

```
;Value 17: (#t #t () () () ())
```

```
1 ]=> (map variable (list 'c1 'c2 'x 'y p5 p7))
```

```
;Value 18: (() () #t #t () ())
```

```
1 ]=> (map pattern (list 'c1 'c2 'x 'y p5 p7))
```

```
;Value 19: (() () () () #t #t)
```

```
1 ]=> (diff 'c1 'c2)
```

```
;Value 20: (c2 c1)
```

```
1 ]=> (diff 'x 'c3)
```

```
;Value 21: (c3 x)
```

```
1 ]=> (diff 'c3 p1)
```

```
;Value 22: ((a b (q y) (a z) w) c3)
```

```
1 ]=> (diff 'y 'v)
```

```
;Value 23: (y v)
```

Recall that

p1 = '(a b (q y) (a z) w))

p2 = '(v b x (w z) w))

p4 = '(v b c (a c) w))

p5 = '(x b c (x v) w))

1]=> (diff p4 p5)

;Value 24: ((a w) x)

1]=> (diff p1 p2)

;Value 25: (a v)

1]=> (define s (unify p1 p2))

;Value: s

1]=> s

;Value 26: ((a w) ((q y) x) (a v))

1]=> (appsub s p1)

;Value 27: (a b (q y) (a z) a)

1]=> (appsub s p2)

;Value 28: (a b (q y) (a z) a)

1]=> (equal? (appsub s p1) (appsub s p2))

;Value: #t

```

p3 = '(v      b      c      (a c)      w) )
p4 = '(x      b      c      (x v)      w) )
p5 = '((a w)    b      c      (v x)      v) )
p6 = '(x      (x x x)  v      (v v v)      ) )
p7 = '((y y y)  z      (z z z)  vZ      ) )

```

```

1 ]=> (unify p3 p4) 1 ]=> (unify p4 p5)
;Value: not-unifiable ;Value: not-unifiable

```

```

1 ]=> (define longsub (unify p6 p7))
;Value: longsub

```

```

1 ]=> (pp longsub)

```

```

((((((y y y) (y y y) (y y y))
  ((y y y) (y y y) (y y y))
  ((y y y) (y y y) (y y y)))
  (((y y y) (y y y) (y y y))
  ((y y y) (y y y) (y y y))
  ((y y y) (y y y) (y y y)))
  (((y y y) (y y y) (y y y))
  ((y y y) (y y y) (y y y))
  ((y y y) (y y y) (y y y))))
vZ)
((((((y y y) (y y y) (y y y))
  ((y y y) (y y y) (y y y))
  ((y y y) (y y y) (y y y)))
  v)
  (((y y y) (y y y) (y y y)) z)
  ((y y y) x))

```