Variables: symbol strings beginning with uppercase characters:

E.g., V3 Uabc Aaa, A8xv, F, Kx17tr, ...

Constants: symbol strings beginning with lowercase characters:

E.g., c1, k, fx, t17, xX27dV

terms:

t1	=	p(a,	b,	f(q,Y),	f(a,Z), W)
t2	=	p(V,	b,	X,	f(W,Z), W)
t3	=	p(V,	b,	c,	f(a,c), W)
t4	=	p(X,	b,	c,	f(X,V), W)
t5	=	p(f(a, W),	b,	c,	f(V,X), V)
t6	=	m(X,	g(X,X,X),	V,	)
t7	=	m(g(Y,Y,Y),	Z,	g(Z,Z,Z))	

The *difference*, diff(A, B), of two expressions A and B is defined to be their first two unequal subexpressions (scanning left to right).

The difference of

h(a, b, f(q,Y), f(a,Z), W)
and h(X, b, c, f(X,V), W)

is <a, X>

The difference of

h(a, b, f(q,Y), f(a,Z), W)
and h(a, b, c, f(X,V), W)

is <f(q Y), c>

The difference of

g(a, b, c, f(V,X), W)
and g(a, b, c, f(V,Y), W)

is <X, Y>

The difference of

r(a, b, c, f(X,V), W)
and r(a, b, c, f(X,V), W)

is <empty>

A difference is *reducible* if one of the subexpressions is a variable **not** occurring in the other.

UNIFYING expressions: can we unify t1 and t2 ?

(Can we make them identical?)

t1	=	(a,	b,	f(q, Y)	f(a, Z),	W)
t2	=	(V,	b,	X,	f(W, Z),	W)

Yes, if:

	V	becomes	a
and	X	becomes	f(q, Y)
and	W	becomes	a

The substitution of expressions for variables is represented as a set of *components*.

We can write each component as a pointer from the variable to the expression (typical implementation).

The variable is bound to the expression; i.e., the expression replaces the variable.

The substitution above would be represented as:

{V --> a, X --> f(q, y), W --> a}

U N I F I C A T I O N (informal algorithm)

To unify expressions A and B, we start with
A, B, diff(A, B), and the empty substitution S = {}.

While diff(A, B) is reducible:

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) = <X, h(Y, Z)>
------------------------------------------------------
A = p(h(Y, Z),   f(g(Y)),  f(h(Y, Z)))
B = p(h(Y, Z),   f(Z    ), f(h(U, V)))

             S = {X --> h(Y,Z)}
(diff A B) = <g(Y), Z>
------------------------------------------------------
```

```
-------------------------------------------------
     (Same as bottom of last slide)
-------------------------------------------------
A = p(h(Y, Z), f(g(Y)), f(h(Y, Z)))
B = p(h(Y, Z), f(Z    ), f(h(U, V)))

         S = {X --> h(Y,Z)}
(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 = {X --> h(Y, g(Y)), Z --> g(Y)}
(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 = {X --> h(Y, g(Y)), Z --> g(Y), U --> Y}
(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 = {X --> h(Y, g(Y)), Z --> g(Y), U --> Y, V --> g(Y)}
(diff A B) = <empty>
-------------------------------------------------
```

LIST NOTATION

We could choose "dot" as a binary function
to take the place of "." in SCHEME.

We could choose "nil" as a constant
to take the place of #f, or (), in SCHEME.

Then [X | T] is syntactic sugar for dot(X,T),

Similarly, [a, b, c]
is syntactic sugar for dot(a, dot(b, dot(c, nil)))
and [] is syntactic sugar for nil.

When Prolog matches [X | T] with [a, b, c],
the binding is {X --> a, T --> [b, c]}

But ask Prolog to match dot(X,T)
with dot(a, dot(b, dot(c, nil)))

The binding is
{X --> a, T --> dot(b, dot(c, nil))}

But this is **still** {X --> a, T --> [b, c]}