Prolog Structures

- Aka, structured or compound objects
- An object with several components.
- Similar to Pascal's Record-type.
- Used to group things together.

functor arguments

course (prolog, chris, mon, 11)

The arity of a functor is the number of arguments.

Structures – Courses

Below is a database of courses and when they meet. Write the following predicates:

- lectures (Lecturer, Day) succeeds if Lecturer has a class on Day.
- duration (Course, Length) computes how many hours Course meets.
- occupied (Room, Day, Time) succeeds if Room is being used on Day at Time.

% course (class, meetingtime, prof, hall).
course (c231, time (mon, 4, 5), cc, plt1).
course (c231, time (wed, 10, 11), cc, plt1).
course (c231, time (thu, 4, 5), cc, plt1).
course (c363, time (mon, 11, 12), cc, slt1).
course (c363, time (thu, 11, 12), cc, slt1).

duration (Course, Length) :-
course (Course, time (Day, Start, Finish), Lec, Loc),
Length is Finish - Start.

occupied (Room, Day, Time) :-
course (Course, time (Day, Start, Finish), Lec, Room),
Start <= Time,
Time <= Finish.
course(c231, time(mon, 4, 5), cc, plt1).
course(c231, time(wed, 10, 11), cc, plt1).
course(c231, time(thu, 4, 5), cc, plt1).
course(c363, time(mon, 11, 12), cc, slt1).
course(c363, time(thu, 11, 12), cc, slt1).

?- occupied(slt1, mon, 11).
yes
?- lectures(cc, mon).
yes

### Binary Trees

We can represent trees as nested structures:

\[
\text{tree} \left( \text{Element}, \text{Left}, \text{Right} \right)
\]

\[
\begin{align*}
\text{tree(s,}
\text{tree(b, void, void),}
\text{tree(x,}
\text{tree(u, void, void), }
\text{void).
\end{align*}
\]

### Binary Search Trees

Write a predicate `member(T, x)` that succeeds if `x` is a member of the binary search tree `T`:

\[
\begin{align*}
\text{atree(}
\text{tree(8,}
\text{tree(4,}
\text{tree(2, void, void),}
\text{tree(7,}
\text{tree(5, void, void), }
\text{void)),}
\text{tree(10,}
\text{tree(9, void, void), }
\text{void))).}
\end{align*}
\]

?- atree(T), tree_member(T, 5).
Binary Trees – Isomorphism

Tree isomorphism:

```
  A
 /|
B C
```

Isomorphic

Two binary trees $T_1$ and $T_2$ are isomorphic if $T_2$ can be obtained by reordering the branches of the subtrees of $T_1$.

Write a predicate `tree_iso(T1, T2)` that succeeds if the two trees are isomorphic.

```
372—Fall 2004—11
```

[9]

Binary Trees – Counting Nodes

Write a predicate `size_of_tree(Tree, Size)` which computes the number of nodes in a tree.

```
size_of_tree(Tree, Size) :-
    size_of_tree(Tree, 0, Size).
```

```
size_of_tree(void, Size, Size).
size_of_tree(tree(_, L, R), SizeIn, SizeOut) :-
    Size1 is SizeIn + 1,
    size_of_tree(L, Size1, Size2),
    size_of_tree(R, Size2, SizeOut).
```

We use a so-called **accumulator pair** to pass around the current size of the tree.

```
372—Fall 2004—11
```

[10]
Write a predicate \( \text{subs}(T_1, T_2, \text{Old}, \text{New}) \) which replaces all occurrences of \( \text{Old} \) with \( \text{New} \) in tree \( T_1 \):

\[
\text{subs}(X, Y, \text{void}, \text{void}).
\]

\[
\text{subs}(X, Y, \text{tree}(X, L_1, R_1), \text{tree}(Y, L_2, R_2)) :-
\text{subs}(X, Y, L_1, L_2),
\text{subs}(X, Y, R_1, R_2).
\]

\[
\text{subs}(X, Y, \text{tree}(Z, L_1, R_1), \text{tree}(Z, L_2, R_2)) :-
X =\not= Y, \text{subs}(X, Y, L_1, L_2),
\text{subs}(X, Y, R_1, R_2).
\]

### Symbolic Differentiation

\[
\frac{dc}{dx} = 0
\]

\[
\frac{dx}{dx} = 1
\]

\[
\frac{d(U^c)}{dx} = cU^{c-1} \frac{dU}{dx}
\]

\[
\frac{d(-U)}{dx} = -\frac{dU}{dx}
\]

\[
\frac{d(U + V)}{dx} = \frac{dU}{dx} + \frac{dV}{dx}
\]

\[
\frac{d(U - V)}{dx} = \frac{dU}{dx} - \frac{dV}{dx}
\]
Symbolic Differentiation...

\[
\frac{dc}{dx} = 0 \\
\frac{dx}{dx} = 1 \\
\frac{d(U^c)}{dx} = cU^{c-1}\frac{dU}{dx}
\]

\[
\text{deriv}(C, X, 0) :- \text{number}(C).
\]
\[
\text{deriv}(X, X, 1).
\]
\[
\text{deriv}(U \^C, X, C \^L \* DU) :- \\
\quad \text{number}(C), L \text{ is } C - 1, \text{deriv}(U, X, DU).
\]

\[
\frac{d(-U)}{dx} = -\frac{dU}{dx}
\]
\[
\frac{d(U + V)}{dx} = \frac{dU}{dx} + \frac{dV}{dx}
\]

\[
\text{deriv}(-U, X, -DU) :- \\
\quad \text{deriv}(U, X, DU).
\]
\[
\text{deriv}(U+V, X, DU + DV) :- \\
\quad \text{deriv}(U, X, DU), \\
\quad \text{deriv}(V, X, DV).
\]

Symbolic Differentiation...

\[
\frac{d(U - V)}{dx} = \frac{dU}{dx} - \frac{dV}{dx}
\]
\[
\frac{d(cU)}{dx} = c\frac{dV}{dx}
\]

\[
\text{deriv}(U-V, X, \underline{\text{_______}}) :- \\
\quad <\text{left as an exercise}>
\]

\[
\text{deriv}(U*V, X, \underline{\text{_______}}) :- \\
\quad <\text{left as an exercise}>
\]

\[
\frac{d(U/V)}{dx} = \frac{U\frac{dV}{dx} - V\frac{dU}{dx}}{V^2}
\]

\[
\text{deriv}(U/V, X, \underline{\text{_______}}) :- \\
\quad <\text{left as an exercise}>
\]
Symbolic Differentiation...

\[
\frac{d(\ln U)}{dx} = \frac{U^{-1} dU}{dx}
\]

\[
\frac{d(\sin(U))}{dx} = \frac{dU}{dx} \cos(U)
\]

\[
\frac{d(\cos(U))}{dx} = -\frac{dU}{dx} \sin(U)
\]

d\ln(U) / dx = \frac{1}{U} \cdot \frac{dU}{dx}

d(sin(U)) / dx = \cos(U) \cdot \frac{dU}{dx}

d(cos(U)) / dx = -\sin(U) \cdot \frac{dU}{dx}

deriv(log(U), X, ______) :- \left\{ \right.

deriv(sin(U), X, ______) :- \left\{ \right.

deriv(cos(U), X, ______) :- \left\{ \right.

?- deriv(x, x, D).
D = 1

?- deriv(sin(x), x, D).
D = \cos(x)

?- deriv(sin(x) + cos(x), x, D).
D = \sin(x) * (-\sin(x)) + \cos(x) * (\cos(x))

?- deriv(1 / x, x, D).
D = (x*0-1*1) / (x*x)

?- deriv(1/sin(x), x, D).
D = (\sin(x)*0-1* (\cos(x)))+(\sin(x)*)sin(x))

?- deriv(x ^3, x, D).
D = 1*3*x^2

?- deriv(x^3 + x^2 + 1, x, D).
D = 1*3*x^2+1*2*x^1+0

?- deriv(3 * x ^ 3, x, D).
D = 3* (1*3*x^2)+x^3*0

?- deriv(4* x ^ 3 + 4 * x^2 + x - 1, x, D).
D = 4* (1*3*x^2)+x^3*0+(4* (1*2*x^1)+x^2*0)+1-0
Readings and References

- Read Clocksin-Mellish, Chapter ???.

Prolog So Far...

- Prolog terms:
  - atoms (a, 1, 3.14)
  - structures
    - guitar(ovation, 1111, 1975)

- Infix expressions are abbreviations of “normal” Prolog terms:

<table>
<thead>
<tr>
<th>infix</th>
<th>prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>a + b</td>
<td>+(a, b)</td>
</tr>
<tr>
<td>a + b* c</td>
<td>+(a, *(b, c))</td>
</tr>
</tbody>
</table>