Introduction

Prolog Structures

- Aka, `structured` or `compound` objects
- An object with several components.
- Similar to Pascal's `Record`-type, C's `struct`, Haskell's `tuples`.
- Used to group things together.
- The `arity` of a functor is the number of arguments.

Example – Course

```prolog
functor course arguments
  course(prolog,chris,mon,11)
```
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 database:

```
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).
```

Predicate definitions:

```
lectures(Lecturer, Day) :-
    course(Course, time(Day,_,_), Lecturer, _).

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.
```

Example queries:

```
?- occupied(slt1, mon, 11).
yes

?- lectures(cc, mon).
yes
```
We can represent trees as nested structures:

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

\[
\text{tree}(s, \\
\quad \text{tree}(b, \text{void}, \text{void}), \\
\quad \text{tree}(x, \\
\quad \quad \text{tree}(u, \text{void}, \text{void}), \text{void}).
\]

Write a predicate \(\text{member}(T, x)\) that succeeds if \(x\) is a member of the binary search tree \(T\):

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

?- atree(T), tree_member(5, T).

Write a predicate \(\text{tree} \_\text{iso}(T_1, T_2)\) that succeeds if the two trees are isomorphic.

Tree isomorphism:

Two binary trees \(T_1\) and \(T_2\) are isomorphic if \(T_2\) can be obtained by reordering the branches of the subtrees of \(T_1\).
Binary Trees – Isomorphism

- tree_isomorphic(void, void).
- tree_isomorphic(tree(X, L1, R1), tree(X, L2, R2)) :- tree_isomorphic(L1, L2), tree_isomorphic(R1, R2).
- tree_isomorphic(tree(X, L1, R1), tree(X, L2, R2)) :- tree_isomorphic(L1, R2), tree_isomorphic(R1, L2).

1. Check if the roots of the current subtrees are identical;
2. Check if the subtrees are isomorphic;
3. If they are not, backtrack, swap the subtrees, and again check if they are isomorphic.

Binary Trees – Counting Nodes

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

- size_of_tree(void, Size).
- size_of_tree(tree(X, L1, R1), SizeIn, SizeOut) :-
  Size1 is SizeIn + 1,
  size_of_tree(L1, Size1, Size2),
  size_of_tree(R1, Size2, SizeOut).

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

Binary Trees – Tree Substitution

- Write a predicate subs(T1, T2, Old, New) which replaces all occurrences of Old with New in tree T1:

  - subs(X, Y, void, void).
  - subs(X, Y, tree(X, L1, R1), tree(Y, L2, R2)) :-
    subs(X, Y, L1, L2),
    subs(X, Y, R1, R2).
  - subs(X, Y, tree(Z, L1, R1), tree(Z, L2, R2)) :-
    X =\= Y, subs(X, Y, L1, L2),
    subs(X, Y, R1, R2).
Symbolic Differentiation

\[
\begin{align*}
\frac{dc}{dx} &= 0 \quad \text{(1)} \\
\frac{dx}{dx} &= 1 \quad \text{(2)} \\
\frac{d(U^c)}{dx} &= cU^{c-1} \frac{dU}{dx} \quad \text{(3)} \\
\frac{d(-U)}{dx} &= -\frac{dU}{dx} \quad \text{(4)} \\
\frac{d(U+V)}{dx} &= \frac{dU}{dx} + \frac{dV}{dx} \quad \text{(5)} \\
\frac{d(U-V)}{dx} &= \frac{dU}{dx} - \frac{dU}{dx} \quad \text{(6)} \\
\frac{d(cU)}{dx} &= c \frac{dU}{dx} \quad \text{(7)} \\
\frac{d(UV)}{dx} &= U \frac{dV}{dx} + V \frac{dU}{dx} \quad \text{(8)} \\
\frac{d(U/V)}{dx} &= \frac{V \frac{dU}{dx} - U \frac{dV}{dx}}{V^2} \quad \text{(9)} \\
\frac{d(lnU)}{dx} &= \frac{dU}{dx} \frac{1}{U} \quad \text{(10)} \\
\frac{d(sin(U))}{dx} &= \frac{dU}{dx} \cos(U) \quad \text{(11)} \\
\frac{d(cos(U))}{dx} &= -\frac{dU}{dx} \sin(U) \quad \text{(12)}
\end{align*}
\]
Symbolic Differentiation...

\[
\frac{dc}{dx} = 0
\]  \hspace{1cm}  (1)

\[
\frac{dx}{dx} = 1
\]  \hspace{1cm}  (2)

\[
\frac{d(U^c)}{dx} = cU^{c-1} \frac{dU}{dx}
\]  \hspace{1cm}  (3)

derv(C, X, 0) :- number(C).

derv(X, X, 1).

derv(U ^C, X, C * U ^L * DU) :-
    number(C), L is C - 1, deriv(U, X, DU).

derv(U - V, X, -DU) :-
    deriv(U, X, DU), deriv(V, X, DV).

derv(U*V, X, _____) :-
    <left as an exercise>

derv(C*U, X, _____) :-
    <left as an exercise>
Symbolic Differentiation...

\[
\frac{d}{dx}(\ln U) = U^{-1} \frac{dU}{dx} \quad (10)
\]

\[
\frac{d}{dx}(\sin(U)) = \frac{dU}{dx} \cos(U) \quad (11)
\]

\[
\frac{d}{dx}(\cos(U)) = -\frac{dU}{dx} \sin(U) \quad (12)
\]

driv(log(U), X, ___) :- \text{<left as an exercise>}

driv(sin(U), X, ___) :- \text{<left as an exercise>}

driv(cos(U), X, ___) :- \text{<left as an exercise>}

Symbolic Differentiation...

\(?- \text{deriv(x, x, D).} \quad D = 1\)

\(?- \text{deriv(sin(x), x, D).} \quad D = \text{1*cos(x)}\)

\(?- \text{deriv(sin(x) + cos(x), x, D).} \quad D = \text{1*cos(x)}+ (-\text{1*sin(x)})\)

\(?- \text{deriv(sin(x) * cos(x), x, D).} \quad D = \text{sin(x)}* (-\text{1*sin(x)}) +\text{cos(x)}* (\text{1*cos(x)})\)

\(?- \text{deriv(1 / x, x, D).} \quad D = (\text{x*0-1*1})/ (\text{x*x})\)

Symbolic Differentiation...

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

\(?- \text{deriv(x^3, x, D).} \quad D = 1*3*x^2\)

\(?- \text{deriv(x^3 + x^2 + 1, x, D).} \quad D = 1*3*x^2+1*2*x^1+0\)

\(?- \text{deriv(3 * x^3, x, D).} \quad D = 3* (1*3*x^2)+x^3*0\)

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

- Read Clocksin-Mellish, Sections 2.1.3, 3.1.

Summary

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>