First-order logic: terms, atoms and quantifiers

Terms

- a constant, variable or functional expression (a function applied to a tuple of terms)

$Expression$	$Term?$
$peter$	$\surd$
X	$\surd$
$log(X)$	$\surd$
$son(peter, peter)$	$\times$
$log(son(peter, peter))$	$\times$
$sin(log(cos(X/2)))$	$\surd$

Atoms

- predicate symbol applied to a tuple of terms ($son(spock, sarek)$)

Arity of function or predicate symbol is the number of terms that each is applied

to. Thus, in $f(a, f(b, Y, Z), q(r(X)))$, the outermost f has arity 1, the inner f has arity 3, q, r have arity 1

– By convention, function and predicate symbols are denoted by $Name/Arity$

Quantifiers

$\forall$ means "for all". It is a way of stating something about all objects in the world without enumerating them. For example, $\forall X\ likes(steve, X)$: $steve$ likes everyone

$\exists$ means "there exists". It is a way of stating the existence of some object in the world without explicitly identifying it. For example, $\exists X\ likes(steve, X)$: $steve$ likes someone

Animals again: monadic predicates

The statement *"Any animal that has hair is a mammal"* can now be written as a clause using monadic predicates (i.e. predicates with arity 1)

$$\forall X \; \mathit{is\_mammal}(X) \leftarrow \mathit{has\_hair}(X)$$

Usually clauses are written without explicit mention of the quantifiers:

$$\mathit{is\_mammal}(X) \leftarrow \mathit{has\_hair}(X)$$

$$\mathit{is\_mammal}(X) \leftarrow \mathit{has\_milk}(X)$$

$$\mathit{is\_bird}(X) \leftarrow \mathit{has\_feathers}(X)$$

$\ldots$

A Datalog "expert" system

Here are the rules with monadic predicates:

```
is_mammal(X) :- has_hair(X).
is_mammal(X) :- has_milk(X).
is_bird(X) :- has_feathers(X).
is_bird(X) :- can_fly(X), has_eggs(X).
is_carnivore(X) :- is_mammal(X), eats_meat(X).
is_carnivore(X) :- has_pointed_teeth(X), has_claws(X), has_pointy_eyes(X).
cheetah(X) :- is_carnivore(X), has_tawny_colour(X), has_dark_spots(X).
tiger(X) :- is_carnivore, has_tawny_colour(X), has_black_stripes(X).
penguin(X) :- is_bird(X), cannot_fly(X), can_swim(X).
```

Now here are some statements particular animals:

```
has_hair(peter).
has_green_eyes(peter).
eats_meat(peter).
has_milk(bob).
has_tawny_colour(bob).
can_fly(bob).
fat(peter).
has_tawny_colour(peter).
has_black_stripes(peter).
eats_meat(bob)
has_dark_spots(bob).
```

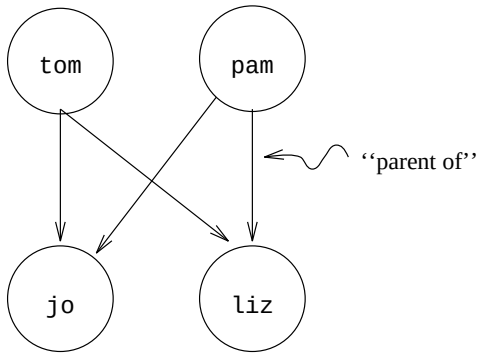
What are the logical consequences of all the clauses?

Monadic predicates: not expressive enough

We can now make statements like *"Every son has a parent"*:
$\forall X \exists Y \; parent(Y) \; \leftarrow \; son(X)$

- But, for more complex relationships, we will need predicates of arity > 1

- Usually, relationships can be described pictorially by a directed acyclic graph (DAG). Here is a parent-child relation:



- The parent-child relation could also

be specified as a set of ordered pairs $< X, Y >$

– Or, as a set of definite clauses

$$parent(tom, jo) \leftarrow$$
$$parent(pam, jo) \leftarrow$$
$$parent(tom, liz) \leftarrow$$
$$parent(pam, liz) \leftarrow$$

Full Datalog: variables, constants and recursion

Consider the *predecessor* relation, namely, all ordered tuples $< X, Y >$ $s.t.$ X is an ancestor of Y. This set will include Y's parents, Y's grandparents, Y's grandparents' parents, etc.

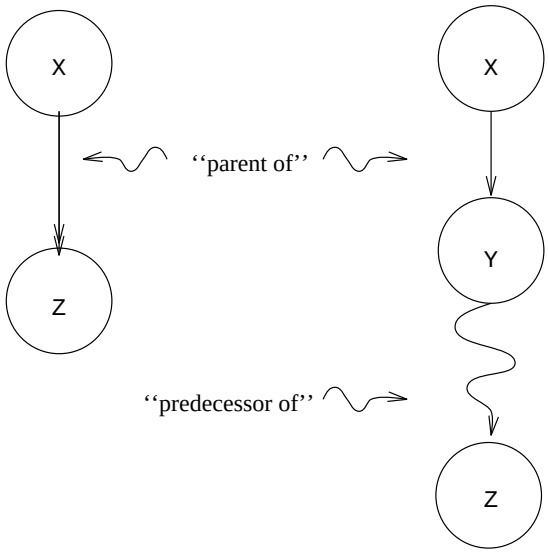
$$
\begin{aligned}
pred(X,Y) &\leftarrow parent(X,Y) \\
pred(X,Z) &\leftarrow parent(X,Y), parent(Y,Z) \\
pred(X,Z) &\leftarrow parent(X,Y1), parent(Y1,Y2), parent(Y2,Z) \\
&\ldots
\end{aligned}
$$

Variables and constants are not enough: we need *recursion*

$\forall X, Z$ X is a predecessor of Z if

1. X is a parent of Z; or
2. X is a parent of some Y, and Y is a predecessor of Z

The *predecessor* relation is thus:



$$pred(X, Y) \leftarrow parent(X, Y)$$
$$pred(X, Z) \leftarrow parent(X, Y), pred(Y, Z)$$

Datalog is not expressive enough

To express arithmetic operations, lists of objects, etc. it is not enough to simply allow variables and constants as terms

– We will need *function* symbols

Consider Peano's postulates for the set of natural numbers $\mathcal{N}$

1. The constant 0 is in $\mathcal{N}$
2. if X is in $\mathcal{N}$ then $s(X)$ is in $\mathcal{N}$
3. There are no other elements in $\mathcal{N}$
4. There is no X in $\mathcal{N}$ $s.t.$ $s(X) = 0$
5. There are no X, Y in $\mathcal{N}$ $s.t.$ $s(X) = s(Y)$ and $X \neq Y$

We can write a definite clause definition for enumerating the elements of $\mathcal{N}$

- 1 constant symbol, 1 unary function symbol

$$natural(0) \leftarrow$$
$$natural(s(X)) \leftarrow natural(X)$$

- They are generated by asking:

natural(N)?

More functions: lists

Lists are simply collections of objects. For e.g. $1, 2, 3 \ldots$ or $1, a, dog, \ldots$

Lists are defined as follows:

1. The constant nil is a list
2. If X is a term, and Y is a list then $.(X, Y)$ is a list

– So the list $1, 2, 3$ is represented as:

$$.(1, .(2, .(3, nil)))$$

– Usually logic programming systems use a "[" "]" notation, in which the constant nil is represented as [] and the list $1, 2, 3$ is $[1, 2, 3]$

– In this notation, the symbol | is used to separate a list into a "head" (the elements to the left of the |) and a "tail" (the list to the right of the |). Thus:

List	Represented as	Values of variables
$[1,2,3]$	$[X\|Y]$	$X = 1, Y = [2,3]$
$[[1,2],3]$	$[X\|Y]$	$X = [1,2], Y = [3]$
$[1]$	$[X\|Y]$	$X = 1, Y = []$
$[1\|2]$	$[X\|Y]$	$X = 1, Y = 2$
$[1]$	$[X,Y]$	
$[1,2,3]$	$[X,Y\|Z]$	$X = 1, Y = 2, Z = [3]$

Predicates + Variables + Constants + Functions

Prolog

Executing a logic program

Consider the following set of clauses S:

$$
\begin{aligned}
&likes(john, flowers) \leftarrow \\
&likes(mary, food) \leftarrow \\
&likes(mary, wine) \leftarrow \\
&likes(john, wine) \leftarrow \\
&likes(john, mary) \leftarrow \\
&likes(paul, mary) \leftarrow
\end{aligned}
$$

– If you entered these clauses into a program capable of executing logic programs, and asked:

$$likes(john, X)?$$

– You will get a number of answers:

$$
\begin{aligned}
X &= flowers \\
X &= wine \\
X &= mary
\end{aligned}
$$

$$likes(john, X), likes(mary, X)?$$

$$X = wine$$

How this works will be examined in detail later. For now, consider *likes(john,X)?*

1. Start search from 1^{st} clause

2. Search for any clause whose head has predicate $likes/2$, and 1^{st} argument is $john$

3. If no clause is found *return* otherwise *goto* 4

4. X is associated ("instantiated") with the 2^{nd} argument of the hea d literal, the clause position marked, and the value associated with X is output

5. Start search from clause marked, and *goto* 2