:- 1 Prolog Introduction

1. Predicates
2. Making Queries
3. Variables
4. Compound Queries
5. Clause Bodies
6. Recursive Predicates
7. Logic Databases

Constraints

5 people live in the five houses in a street. Each has a different profession, animal, favorite drink, and each house is a different color.

The Englishman lives in the red house

The Norwegian lives in the first house on the left

The green house is on the right of the white one

The fox is in a house next to the doctor

The horse is in a house next to the diplomat

The Norwegians house is next to the blue one

The owner of the green house drinks coffee

The Spaniard owns a dog

The Japanese is a painter

The Italian drinks tea

Milk is drunk in the middle house

The violinist drinks fruit juice

The sculptor breeds snails

The diplomat lives in the yellow house

Who owns the zebra? Who drinks water?

Real world staff rostering problems have many constraints just like this.

New Abstractions for Logic Programming

- Declarative programming
- Relational programming, table-driven programming
- Unification, partial data structures
- Programming by constraint, reversible programs
- Nondeterminism (and backtracking)

1.1 Predicates

In most programming languages, the basic abstraction is the *function* or perhaps *method*

In Prolog, the basic abstraction is the **predicate** or **relation**

A **function** is a mapping from some inputs to some outputs

A **predicate** instead indicates whether or not some entities are related in a particular way

In logic programming, may test whether some entities are related in a particular way, or to find entities that are related to other entities

Predicates (2)

Here is a simple Prolog program giving information about the British royal family:

```
parent(elizabeth, charles).
parent(diana, william).
parent(diana, harry).
parent(philip, charles).
parent(charles, william).
parent(charles, harry).
```

Here `parent` is a predicate

Each line above is a separate **clause**

Each clause is a separate true statement: should be able to understand each clause separately

A Prolog program may consist of many files, each defining many predicates, each defined by many clauses

1.2 Making Queries

We can use this program to ask about family relationships

Most Prolog systems, such as the SWI Prolog system we will be using, provide an interactive environment

Allows loading and testing code

Prolog prompt is `?-`

Load source file by wrapping file name in single quotes and square brackets, followed by full stop

Usual extension for Prolog source code is `.pl`

Pose queries by typing them at the prompt; Prolog will answer them with `Yes` or `No`

Making Queries (2)

For these lectures, boxed text like this represents terminal sessions

```
toaster% pl
Welcome to SWI-Prolog (Version 4.0.1)
Copyright (c) 1990-2000 University of Amsterdam.

?- ['royals.pl'].
% royals.pl compiled 0.00 sec, 1,200 bytes
Yes
?- parent(diana, harry).
Yes
?- parent(elizabeth, diana).
No
```

1.3 Variables

We can query of whom Elizabeth is parent using a variable:

```
?- parent(elizabeth, X).
X = charles
Yes
```

Prolog variables begin with a *capital letter* or an *underscore*, and follow with letters, digits, and underscores

N.B. Prolog variables are very different than variables in other programming languages

Like Haskell: you cannot change the value of a variable once it is set

Read query as: "find `X` such that `elizabeth` is parent of `X`"

Variables (2)

Some queries have more than one answer:

```
?- parent(diana, X).
X = william ;
X = harry ;
No
```

When Prolog prints an answer, it waits to see if you want more answers

Hit semicolon to ask for more answers; hit return if no more answers are needed

After asking for another answer after `harry`, Prolog says `No`, meaning there are no more answers

1.3.1 Multiple Directions

We could run this query the other way around:

```
?- parent(X, william).
X = diana ;
X = charles ;
No
```

"Find `X` such that `X` is parent of `william`"

This is possible because our program is *declarative*: it specifies information without specifying how it must be used

Well-written logic programs exhibit this flexibility

This is not always practical, as we will see later

Multiple Directions (2)

In general, for *each* argument of a predicate we may pass a variable or a value

Either way, Prolog finds all matching solutions

Think of the values specified as constraining the solutions, while variables do not impose a constraint

A query with only variables imposes no constraints:

```
?- parent(X, Y).
X = elizabeth
Y = charles ;
X = diana
Y = william
```

⋮

1.4 Compound Queries

Prolog queries can invoke multiple predicates, separating them with commas

If variables appear more than once, all occurrences must have same value (as in Algebra)

```
?- parent(elizabeth,X), parent(X,Y).
X = charles
Y = william ;
X = charles
Y = harry ;
No
```

1.4.1 More Predicates

Our program so far includes no information about gender; add it now:

```
female(elizabeth).
female(diana).
male(philip).
male(charles).
male(william).
male(harry).
```

`male` and `female` are predicates defining the gender of the royals

Predicates can take any number of arguments

The number of arguments is called the **arity**

A predicate with only one argument essentially defines a property that certain entities may have

:- Exercise: Motherhood

Give a Prolog query that determines the mother of `william`

1.4.2 Constraints

A good way to think of compound queries is that each part of the query imposes a **constraint** on the solution

Prolog will only show solutions that satisfy all the constraints

Some constraints have no solutions

In general, constraints can be placed in any order

Some orders may be more efficient than others

Some orders may not work as expected; more later

:- Constraints (2)

```
?- parent(X,Y), parent(Y,Z), male(Z), female(X).
X = elizabeth
Y = charles
Z = william ;
X = elizabeth
Y = charles
Z = harry ;
No
?- parent(X,Y), parent(Y,X).
No
?- male(X), female(X).
No
```

:- 1.5 Clause Bodies

Predicates can be defined in terms of other predicates

A clause may have two parts: a **head** and a **body**

Head and body are separated by `:-` (read it as "if")

Head looks like a predicate with arguments, some of which may be variables

Body looks like a query

Means head is true if body is true

Omit `:-` if body is empty (as in previous examples)

Clause with no body is called a **unit clause** or a **fact**

1.5.1 Extending the Example

We can define a grandparent as a parent's parent:

```
grandparent(G, C) :-
        parent(G, P),
        parent(P, C).
```

Read: "`G` is grandparent of `C` if `G` is parent of `P` and `P` is parent of `C`"

Can use this to find grandparents or grandchildren:

```
?- grandparent(elizabeth, X).
X = william ;
X = harry ;
No
?- grandparent(X, william).
X = elizabeth ;
X = philip ;
No
```

:- Exercise: Family Relationships

1. Define a predicate `father(Dad,Kid)` that holds when `Dad` is father of `Kid`
2. Define a predicate `greatgrandparent(Adult,Child)` that holds when `Adult` is a great grandparent of `Child`

1.6 Recursive Predicates

Predicates can also be defined recursively:

```
ancestor(A, A).
ancestor(A, D) :-
        parent(A, C),
        ancestor(C, D).
```

The first clause says that everyone is their own ancestor — not the usual definition, but common in computer science

The second clause says that the parents of your ancestors are your ancestors

This code can be used both for finding ancestors and descendants

1.6.1 Querying Recursive Predicates

```
?- ancestor(A,harry).
A = harry ;
A = elizabeth ;
A = diana ;
A = philip ;
A = charles ;
No
?- ancestor(elizabeth, D).
D = elizabeth ;
D = charles ;
D = william ;
D = harry ;
No
```

1.7 Disjunction

Disjunction in Prolog uses the ; operator, *e.g.*:

```
person(X) :-
        ( male(X) ; female(X) ).
```

Could also have been written:

```
person(X) :- male(X).
person(X) :- female(X).
```

1.8 Logic Databases

Programming by constraint is good for database-like applications (this is similar to the way relational databases work)

```
state(tas).     state(vic).     state(nsw).     state(nt).
state(sa).      state(act).     state(qld).     state(wa).

borders1(vic, nsw).     borders1(vic, sa).    borders1(nt, wa).
borders1(nsw, qld).     borders1(nsw, sa).    borders1(sa, nt).
borders1(qld, sa).      borders1(qld, nt).    borders1(sa, wa).
borders1(act, nsw).

borders(S1, S2) :- borders1(S1, S2).
borders(S1, S2) :- borders1(S2, S1).

capital(tas, hobart).   capital(vic, melbourne).
capital(nsw, sydney).   capital(sa, adelaide).
capital(act, canberra). capital(qld, brisbane).
capital(nt, darwin).    capital(wa, perth).
```

1.8.1 Database Queries

```
?- borders(vic, State), capital(State, Capital).
State = nsw
Capital = sydney ;
State = sa
Capital = adelaide ;
No
?- borders(S, S1), borders(S, S2), S1 \= S2.
S = vic
S1 = nsw
S2 = sa ;
S = vic
S1 = sa
S2 = nsw ;
S = nsw
S1 = qld
S2 = sa
```

N.B. `S1 \= S2` means `S1` is different from `S2`

1.8.2 Not Equals

The \= predicate must be *after* its variables are fixed.

```
?- S1 \= S2, borders(S, S1), borders(S, S2).
No
```

This fails because `S1` and `S2` could be equal at the time the `S1 \= S2` goal is executed

Conjunction is supposed to be commutative, but it isn't; this is one of Prolog's foibles. Explanation later!

Note also that `S1 \= S2` is the same as `\=(S1, S2)`

Prolog syntax supports "operators" (infix, prefix and postfix); several are pre-defined and you can define your own, eg `is_parent_of`, `borders`