PROLOG (*PROGRAMMING IN LOGIC*)

Artificial Intelligence Introduction to Prolog

- Exercises Teaching Material
  - Learn Prolog Now! https://sites.google.com/site/prologsite/home
  - SWI Prolog interpreter http://www.swi-prolog.org/
  - Book: The Art of Prolog by Leon Sterling and Ehud Shapiro.

SWI Prolog

- *Sociaal-Wetenschappelijke Informatica* ("Social Science Informatics")
- Freely available Prolog interpreter
- Works with
  - Linux,
  - Windows, or
  - Mac OS
- There are many more Prolog interpreters

Lecture 1

- Theory
  - Introduction to Prolog
  - Facts, Rules and Queries
  - Prolog Syntax

Aim of this lecture

- Give some simple examples of Prolog programs
- Discuss the three basic constructs in Prolog:
  - Facts
  - Rules
  - Queries
- Introduce other concepts, such as
  - the role of logic
  - unification with the help of variables
- Begin the systematic study of Prolog by defining terms, atoms, and variables

Prolog

- "Programming with Logic"
- Declarative
- Very different from other (procedural) programming languages
- Good for knowledge-rich tasks

Basic idea of Prolog

- Describe the situation of interest
- Ask a question
- Prolog logically deduces new facts about the situation we described
- Prolog gives us its deductions back as answers

Consequences

- Think declaratively, not procedurally
  - Challenging
  - Requires a different mindset
- High-level language
  - Not as efficient as, say, C
  - Good for rapid prototyping
  - Useful in many AI applications

Logic Programs

- A logic program is a set of axioms, or rules, defining relations between objects.
- There are 3 basic constructs which define logic programming.

1. Facts
2. Queries
3. Rules

Facts

- The simplest kind of statement is called a fact.
- Relation holds between objects → Facts
- **Example**: **father(abraham, isaac).**
- This fact says that **Abraham** is the father of **Isaac**, or that the relation father holds between the individuals named **abraham** and **isaac**.

- **Predicate**: Another name for a relation is a predicate.
- **Atoms**: Names of individuals are known as atoms. Example: **abraham** and **isaac.**
- **Program:** A finite set of facts constitutes a program.

Queries

- Queries are a means of retrieving information from a logic program.
- Example: father(**abraham**, **isaac**)?
- Given the facts of previous slide, the answer to this query is **true/yes**.

Atoms

- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, **starting with a lowercase letter.**
  - *Examples*: **abraham**, **big_kahuna_burger**, **hamSandwich etc.**

- An arbitrary sequence of characters enclosed in single quotes
  - *Examples*: **'Vincent'**, **'Suarez's bite'**, **'@$%'**

- A sequence of special characters
  - *Examples*: **: , ; . :-**

Numbers

- 0-9
- Integers: 12, -34, 22342
- Floats: 34573.3234

VARIABLES

- A sequence of characters of upper-case letters, lower-case letters, digits, or underscore, ***starting with*** either an **uppercase letter** or an **underscore.**

- Examples:

  X, Y, Variable, Vincent, _tag etc.

The Logical Variable, Substitutions, and Instances

- Suppose we want to know of whom **abraham** is the father.
- father(abraham, lot)?
- father(abraham, milcah)?
- ,…,
- father (abraham, isaac)? until an answer **yes** is given.
- Better way to use varialbe logically ☺
- father (abraham, **X**)?
- **X=isaac.**

Terms

- Variables. Example: X, Y, etc.
- Constants Example: abraham, isaac, etc.

Compound Terms: (functor)

- A functor is characterized by its name, which is an **atom**, and its **arity**, or number of arguments.
- Example: f(t1,t2,...,tn)
- Atom → f
- Arity → n
- ti are arguments.

Ground Vs. Nonground

- queries, goals or terms without variable is ground.
- Example: foo(a, b) is ground.

- queries, goals or terms with variable is a ground.
- Example: bar(X) is nonground.

Prolog and Logic

- Clearly Prolog has something to do with logic
- Operators
  - Implication :-
  - Conjunction ,
  - Disjunction ;
- Use of modus ponens
- Negation

Rules

- Rules are statements of the form:

  A :- B1,B2,...,Bn. where n >= 0.

- Note that a fact is just a special case of a rule when n = 0. Facts are also called unit clauses.
- Example: A rule expressing the son relationship is

  son(X,Y) :- father(Y,X), male(X).

- Similarly one can define a rule for the daughter relationship:

  daughter(X,Y) :- father(Y,X), female(X).

Knowledge Base

- A collection of facts and rules is called a knowledge base (KB) (or a database) and Prolog programming is all about writing knowledge bases.

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
```

KNOWLEDGE BASE 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?-
```

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- woman(mia).
yes
?- playsAirGuitar(jody).
yes
?- playsAirGuitar(mia).
no
```

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- tattoed(jody).
```

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- tattoed(jody).
no
?-
```

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- tattoed(jody).
ERROR: predicate tattoed/1 not defined.
?-
```

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- party.
```

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- party.
yes
?-
```

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- rockConcert.
```

Knowledge Base 1

```
woman(mia).
woman(jody).
woman(yolanda).
playsAirGuitar(jody).
party.
```

```
?- rockConcert.
no
?-
```

Knowledge Base 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

KNOWLEDGE BASE 2

fact

happy(yolanda).

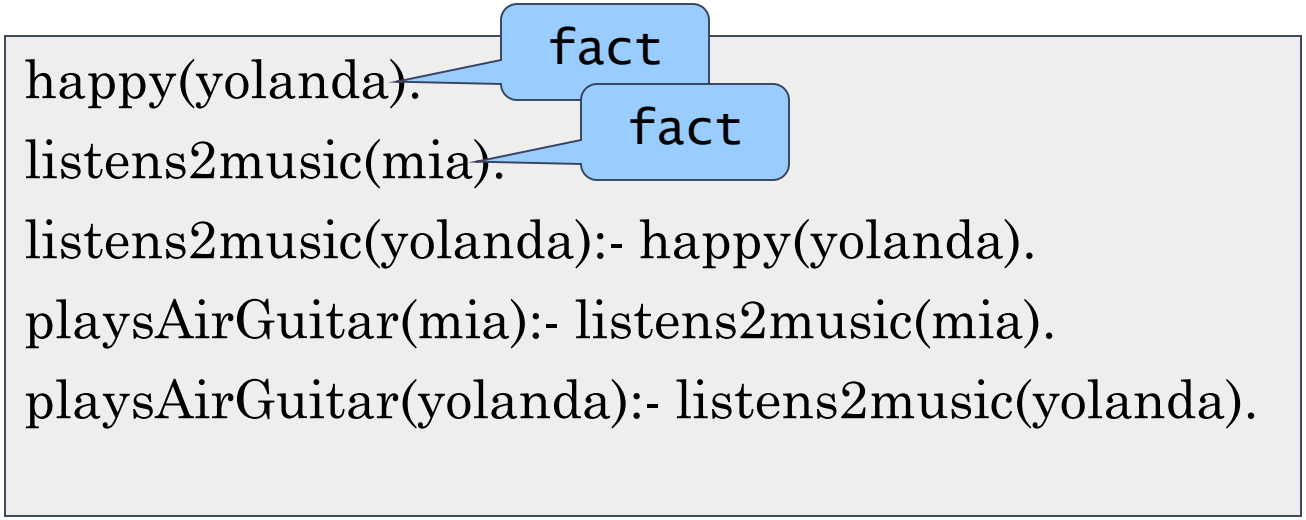
listens2music(mia).

listens2music(yolanda):- happy(yolanda).

playsAirGuitar(mia):- listens2music(mia).

playsAirGuitar(yolanda):- listens2music(yolanda).

KNOWLEDGE BASE 2



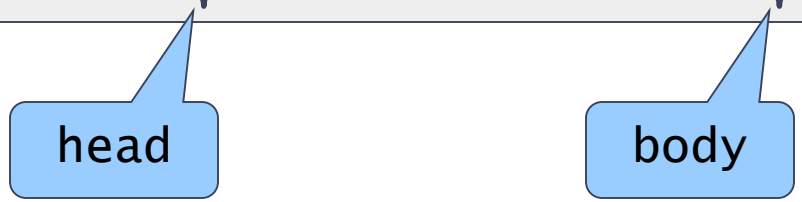
KNOWLEDGE BASE 2

KNOWLEDGE BASE 2

KNOWLEDGE BASE 2

Knowledge Base 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```



Knowledge Base 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?-
```

KNOWLEDGE BASE 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?- playsAirGuitar(mia).
yes
?-
```

KNOWLEDGE BASE 2

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

```
?- playsAirGuitar(mia).
yes
?- playsAirGuitar(yolanda).
yes
```

Clauses

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

There are five clauses in this knowledge base: two facts, and three rules.

The end of a clause is marked with a full stop.

Predicates

```
happy(yolanda).
listens2music(mia).
listens2music(yolanda):- happy(yolanda).
playsAirGuitar(mia):- listens2music(mia).
playsAirGuitar(yolanda):- listens2music(yolanda).
```

*There are three **predicates** in this knowledge base:*
happy, listens2music, and playsAirGuitar

Knowledge Base 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent),
    happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

Expressing Conjunction

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

The comma "," expresses conjunction in Prolog

Knowledge Base 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

```
?- playsAirGuitar(vincent).
no
?-
```

KNOWLEDGE BASE 3

```
happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch).
playsAirGuitar(butch):- listens2music(butch).
```

```
?- playsAirGuitar(butch).
yes
?-
```

Expressing Disjunction

happy(vincent).

listens2music(butch).

playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).

playsAirGuitar(butch):- happy(butch).

playsAirGuitar(butch):- listens2music(butch).

happy(vincent).
listens2music(butch).
playsAirGuitar(vincent):- listens2music(vincent), happy(vincent).
playsAirGuitar(butch):- happy(butch); listens2music(butch).

Knowledge Base 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

Prolog Variables

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
```

Variable Instantiation

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia
```

Asking Alternatives

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia;
```

Asking Alternatives

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia;
X=jody
```

Asking Alternatives

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia;
X=jody;
X=yolanda
```

Asking Alternatives

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- woman(X).
X=mia;
X=jody;
X=yolanda;
no
```

KNOWLEDGE BASE 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- loves(marsellus,X), woman(X).
```

Knowledge Base 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- loves(marsellus,X), woman(X).
X=mia
yes
?-
```

KNOWLEDGE BASE 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- loves(pumpkin,X), woman(X).
```

Knowledge Base 4

```
woman(mia).
woman(jody).
woman(yolanda).

loves(vincent, mia).
loves(marsellus, mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).
```

```
?- loves(pumpkin,X), woman(X).
no
?-
```

KNOWLEDGE BASE 5

```
loves(vincent,mia).
loves(marsellus,mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

KNOWLEDGE BASE 5

```
loves(vincent,mia).
loves(marsellus,mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

```
?- jealous(marsellus,W).
```

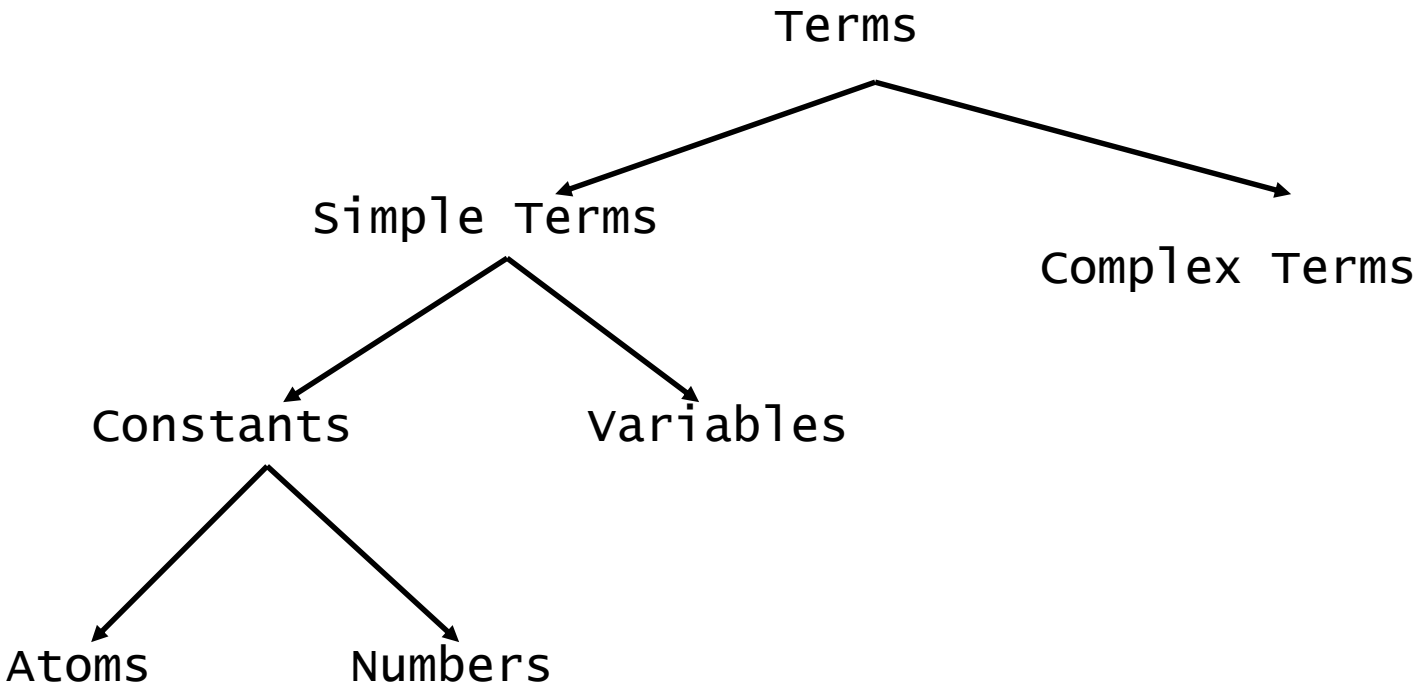
KNOWLEDGE BASE 5

```
loves(vincent,mia).
loves(marsellus,mia).
loves(pumpkin, honey_bunny).
loves(honey_bunny, pumpkin).

jealous(X,Y):- loves(X,Z), loves(Y,Z).
```

```
?- jealous(marsellus,W).
W=vincent
?-
```

Prolog Syntax



Arity

- The number of arguments a complex term has is called its **arity**

- Examples:

woman(mia) is a term with arity 1
loves(vincent,mia) has arity 2
father(father(butch)) arity 1

Arity is important

- In Prolog you can define two predicates with the same functor but with different arity
- Prolog would treat this as two different predicates
- In Prolog documentation arity of a predicate is usually indicated with the suffix "/" followed by a number to indicate the arity

Example of Arity

happy(yolanda).

listens2music(mia).

listens2music(yolanda):- happy(yolanda).

playsAirGuitar(mia):- listens2music(mia).

playsAirGuitar(yolanda):- listens2music(yolanda).

- This knowledge base defines
  - happy/1
  - listens2music/1
  - playsAirGuitar/1

Operators and Functions

- Normal Prolog operators are prefix:
  - @>(Item1, Item2).
  - @=<(Item1, Item2).
  - ==(Item1, Item2).
  - \==(Item1, Item2).
- Some symbols can be used infix: arithmetic and comparison

Comparison Operators

Operator	Meaning
\+	Not (negation by failure)
=, \=, =..	Unification, not unifiable, list unification
==, \==	Term identical, term not identical
@<, @=<, @>, @>=	Term less than, term less than or equal to, etc.
is	Unify left-hand side with result of evaluating right hand side.
=:=, =\=	Arithmetic equal, arithmetic not equal
<, =<, >, >=	Arithmetic less than, etc.
:	
+, -, /\, \/	Add, subtract, bitwise AND, bitwise OR
*, /, //,	Mult, div, integer division
rem, mod	Different versions of mod (see manual)
>>, <<	Shift right, shift left
**, ^. \	Exponentiation, bitwise XOR, bitwise NOT

Arithmetic functions

`sqrt(E)`	square root of *eval*(E)
`atan(E)`	arc tangent of *eval*(E)
`cos(E)`	cosine of *eval*(E)
`acos(E)`	arc cosine of *eval*(E)
`sin(E)`	sine of *eval*(E)
`asin(E)`	arc sine of *eval*(E)
`exp(E)`	*e* raised to the power of *eval*(E)
`log(E)`	natural logarithms of *eval*(E)
`float(E)`	the floating point number equal to *eval*(E)
`ceiling(E)`	rounds *eval*(E) upward to the nearest integer
`floor(E)`	rounds *eval*(E) downward to the nearest integer
`round(E)`	rounds *eval*(E) to the nearest integer
`truncate(E)`	the integer value of *eval*(E)

Arithmetic

- Example.

```
  bonus(Number) :- Number is 2 + 3.
?- bonus(3).
No
?- bonus(5).
Yes
?- bonus(X).
X = 5
```

Arithmetic

- Example.

```
| ?- X is 5 + 2.
X = 7
yes
| ?- X = 5 + 2.
X = 5+2
yes
| ?- X is 5.3 + 7.
X = 12.3
yes
| ?-
```

= and **is** have different meanings

= means term assignment
is means arithmetic assignment

Arithmetic

- Example: geography.pl

```
/*
  north latitudes and west longitudes are positive.
  sound latitudes and east longitudes are negative.
*/
location(tokyo, 35, -139).
location(rome, 41, -12).
location(london, 51, 0).
location(canberra, -31, -149).
location(madrid, 48, 3).
north_of(X, Y) :-
        location(X, Lat1, _),
        location(Y, Lat2, _),
        Lat1 > Lat2.
west_of(X, Y) :-
        location(X, _, Long1),
        location(Y, _, Long2),
        Long1 > Long2.
```

Try:

```
north_of(madrid, tokyo).

west_of(X, tokyo).
west_of(london, X).
```

TRACING

- The prolog command is `trace`

```
|?-trace.
yes
[trace] 1 ?- north_of(madrid,tokyo).
   Call: (7) north_of(madrid, tokyo) ? creep
   Call: (8) location(madrid, _G2207, _G2208) ? creep
   Exit: (8) location(madrid, 48, 3) ? creep
   Call: (8) location(tokyo, _G2207, _G2208) ? creep
   Exit: (8) location(tokyo, 35, -139) ? creep
   Call: (8) 48>35 ? creep
   Exit: (8) 48>35 ? creep
   Exit: (7) north_of(madrid, tokyo) ? creep
true.
For more: visit:
     http://www.cse.unsw.edu.au/~billw/dictionaries/prolog/tracing.html
```

TRACING

The command to turn off is `notrace`

```
|?-notrace.
The debugger is switched off
Yes
|?-
```

Making Decisions

- There are no *if* or *case* statements in prolog.
- How do we make decisions?
- Example: determine how hot it is:

```
// determine how hot it is
void howHot(string &HowHot, int Temp){
   if (Temp >= 100)
         HowHot = "very";
   else if (Temp >= 90)
         HowHot = "pretty";
   else if (Temp >= 70)
         HowHot = "perfect";
   else if (Temp < 70)
         HowHot = "cold";
}
```

Making Decisions

- Making decisions (howHot.pl)

```
% determine how hot it is
Hot(HowHot, Temp) :-
   Temp >= 100,
   HowHot = 'very';
   Temp < 100,
   Temp >= 90,
   HowHot = 'pretty';
   Temp < 90,
   Temp >= 70,
   HowHot = 'perfect';
   Temp < 70,
   HowHot = 'cold'.
```

Can ask
`howHot(X, 80).`
But *not*
`howHot(very, X).`

Cannot use *is* must use =
is only works on arithmetic expressions

Must have this test because prolog will view each *or* clause independently. If we left out Temp < 100 then a temp of 110 would return both "very" and "pretty"

Thanks

?