Declarative Programming
Prolog

CS360

Terms

- Numerical literals
- String literals
  - By default, any string that starts is all lowercase
  - Use single quotes for anything else
- Atoms
  - Essentially, strings
  - Also, name of functors

Usage Details

- Use "." to end a statement
- Use "," to separate clauses, args, etc.
- ctrl-D to exit (or, finish input)
- `consult(` *`file`* `)` (or `[`*`file`*`]`) to read a file
  - Put filename in single quotes
  - `consult( user )` (or `[user]`) to read stdin

Some Handy Predicates

- `working_directory/1`
- `change_directory/1`
- `trace , notrace`
- `listing/1`

Compound Terms

- Look like function calls
  - They’re not
  - Just statements
- Called *predicates*
  - **Only** returns true or false
- Attempts to check the verity of a predicate using *resolution*

Common Operators

- Relational (on numbers):

```
=:=   =\=   <   =<   >   >=
```

- More general relational:

```
  ==   \==   @<   @=<   @>   @>=
```

- These are **not** predicates:

```
  +   -   *   /   **   //
```

- Syntactic Sugar:
  - `+( a, b )` is the same as `a + b`

As a Calculator

- Does **not** return the value of an expression:

```
   | ?- 2 + 3.
   uncaught exception:
error(existence_error(procedure,
(+)/2),top_level/0)
```

- Use the `is` predicate
  - Only verifies a statement:

```
   | ?- is(5, 2+3).
   yes
```

Unification

- Binding variables to values
- (Variables are any string that starts w/a capital letter or "_")

```
| ?- is( X, 5+12 ).
X = 17
yes
```

A Better Example (parents.pl)

- Let’s define some facts:

```
| ?- [user].

compiling user for byte code...

parent( kim, holly ).

parent( margaret, kim ).

parent( margaret, kent ).

parent( esther, margaret ).

parent( herbert, margaret ).

parent( herbert, jean ).
```

Simple Queries

- We enter a query, or goal:

```
| ?- parent( esther, margaret ).
```

- Prolog looks through the DB, in order, to find a fact that matches the query. Found, so

```
yes
```

- If it fails to find one:

```
   | ?- parent( esther, snoopy ).
   no
```

Variables

- Prolog looks for bindings to prove a given query
- Find all of Margaret’s children:

```
 | ?- parent( margaret, X ).
X = kim ? ;
X = kent
yes
```

More Prolog Details - ;

- Use the semicolon to see the next binding prolog finds to resolve the query
- Hit [enter] to stop looking:

```
 | ?- parent( margaret, X ).
X = kim ?
yes
```

Look for Parents

- Find all of Margaret’s parents:

```
| ?- parent( X, margaret ).
X = esther ? ;
X = herbert ? ;
no
```

Conjunction of Terms

- Query can be a list of terms
  - Treated as a logical conjunction of the terms
- Find all who have both Esther and Herbert as parents:

```
   | ?- parent( esther, X ), parent( herbert,
X ).
   X = margaret ? ;
   no
```

  - Note, X is the same variable in both terms
    - “_” is special; a wildcard. Each instance is bound independently

Horn Clauses

- Statements of the form:

  B :- A1, A2, … , An.

  - *B* is called the *head*
  - *A1, A2, … , An* is the *body*
- A clause w/out a body is always true
  - *Axiom*, or *fact*
- A clause w/out a head is a *query* or *goal*

More Conjunction

- Find all of Kim’s grandparents:

```
| ?- parent( G, P ), parent( P, kim ).

G = esther
P = margaret ? ;

G = herbert
P = margaret ? ;

G = tom
P = dave

(15 ms) yes
```

Rules

- We can create more interesting *rules*, to prove more complex relations:

```
  | ?- [user].
  compiling user for byte code...
  ancestor( A, D ) :- parent( A, X ),
ancestor( X, D ).
  ancestor( A, A ).
```

GCD

```
gcd(U,0,U).
gcd(U,V,W) :- V=\=0, R is U mod V,
gcd(V,R,W).
```

```
  | ?- gcd( 15, 6, X ).
  X = 3 ?
  | ?- gcd( X, 6, 3 ).
  uncaught exception:
error(instantiation_error,(is)/2)
```

Lists

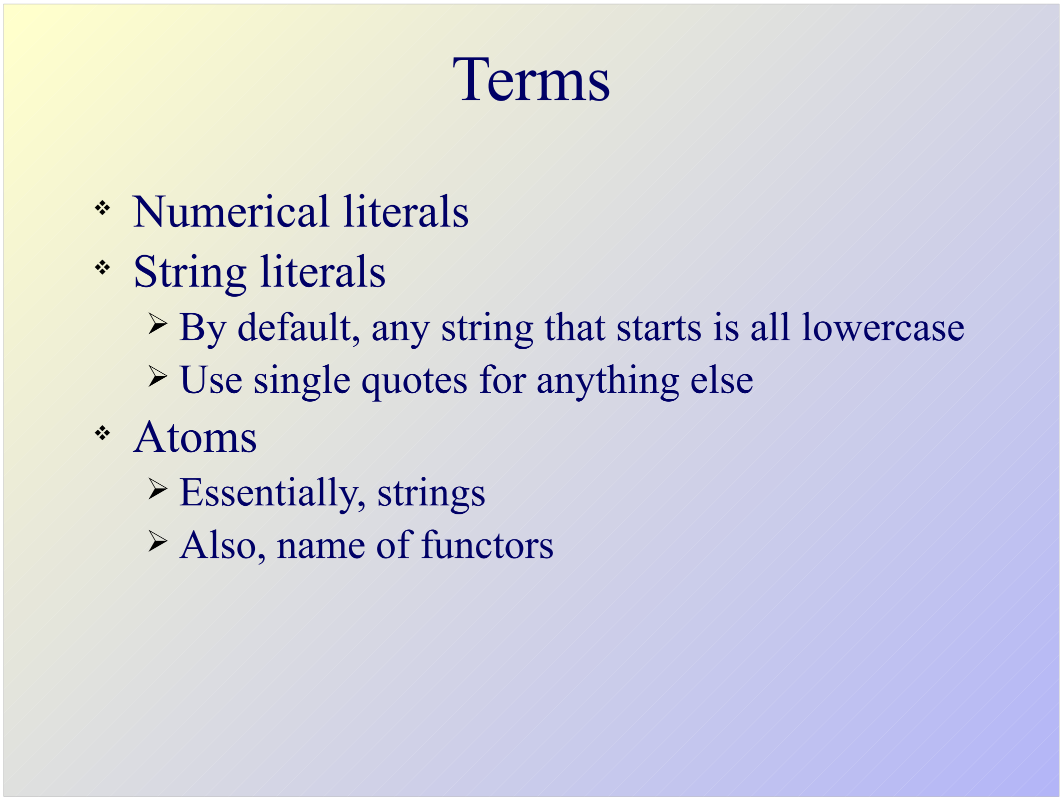
- [] – empty list
- [1, 2, 3] – finite list
- [X | Y]
  - X is the first element
  - Y is the rest

Examples

- member.pl
- lookup.pl
- append.pl
- sort.pl

Declarative Programming
Prolog

CS360

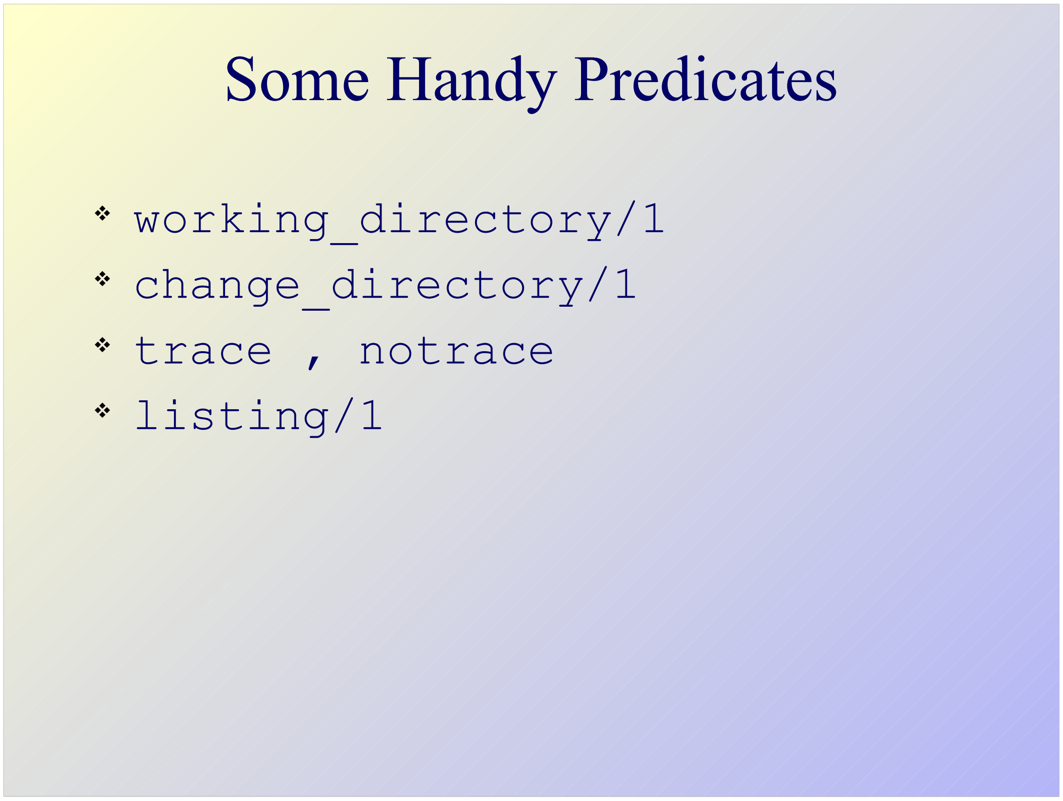


Terms

- Numerical literals
- String literals
  - By default, any string that starts is all lowercase
  - Use single quotes for anything else
- Atoms
  - Essentially, strings
  - Also, name of functors

Usage Details

- Use “.” to end a statement
- Use “,” to separate clauses, args, etc.
- ctrl-D to exit (or, finish input)
- `consult( file )` (or `[file]`) to read a file
  - Put filename in single quotes
  - `consult( user )` (or `[user]`) to read stdin

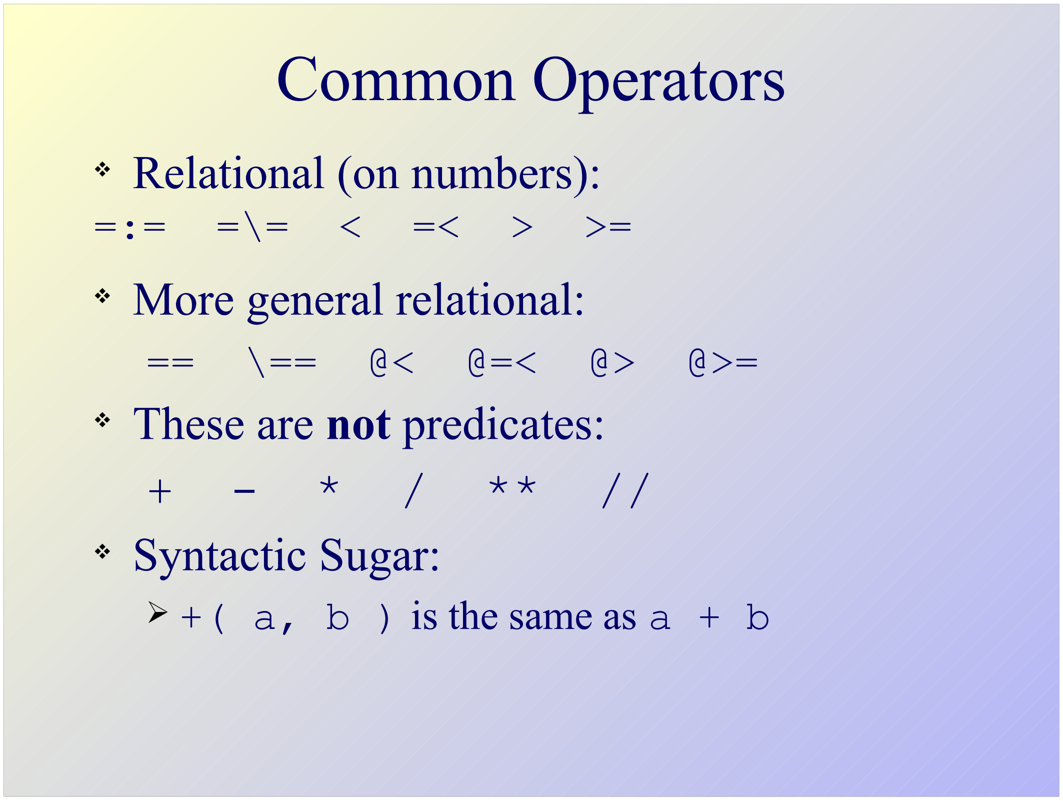


Some Handy Predicates

- `working_directory/1`
- `change_directory/1`
- `trace , notrace`
- `listing/1`

Compound Terms

- Look like function calls
  - They're not
  - Just statements
- Called *predicates*
  - **Only** returns true or false
- Attempts to check the verity of a predicate using *resolution*



Common Operators

- Relational (on numbers):

```
=:=   =\=   <   =<   >   >=
```

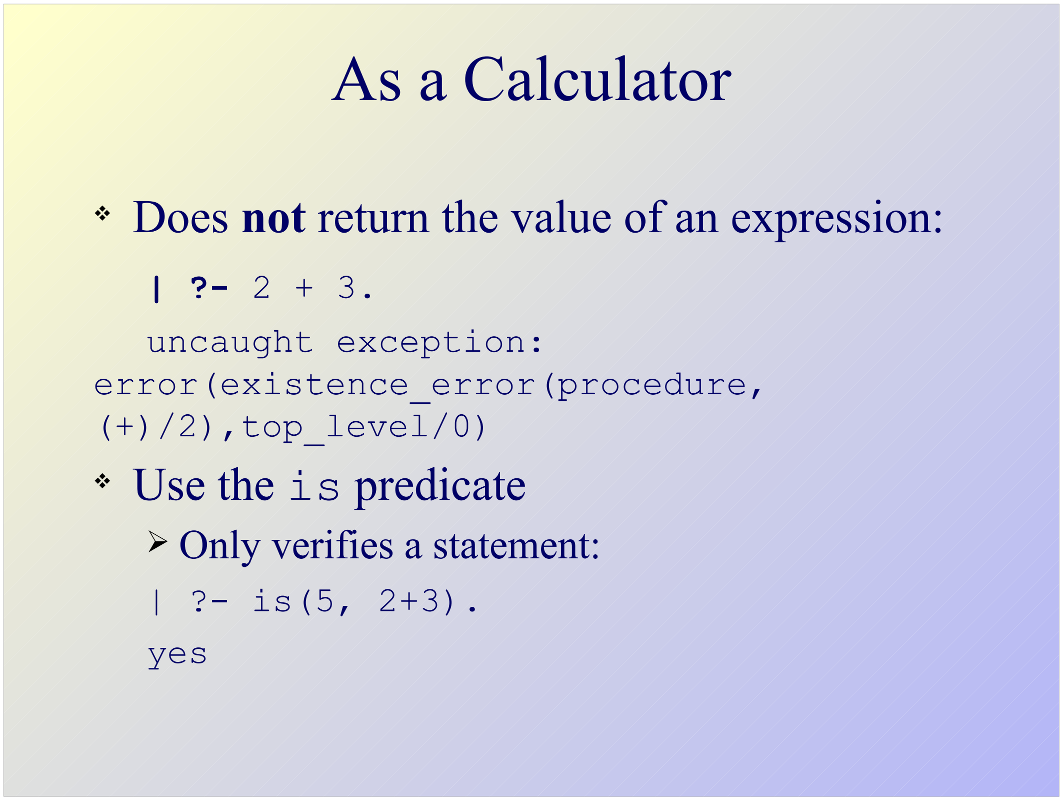
- More general relational:

```
  ==   \==   @<   @=<   @>   @>=
```

- These are **not** predicates:

```
  +   -   *   /   **   //
```

- Syntactic Sugar:
  - `+( a, b )` is the same as `a + b`



As a Calculator

- Does **not** return the value of an expression:

```
   | ?- 2 + 3.
   uncaught exception:
error(existence_error(procedure,
(+)/2),top_level/0)
```

- Use the `is` predicate
  - Only verifies a statement:

```
   | ?- is(5, 2+3).
   yes
```

Unification

- Binding variables to values
- (Variables are any string that starts w/a capital letter or “_”)

```
| ?- is( X, 5+12 ).
X = 17
yes
```

A Better Example (parents.pl)

- Let’s define some facts:

```
| ?- [user].

compiling user for byte code...

parent( kim, holly ).

parent( margaret, kim ).

parent( margaret, kent ).

parent( esther, margaret ).

parent( herbert, margaret ).

parent( herbert, jean ).
```

Simple Queries

- We enter a query, or goal:

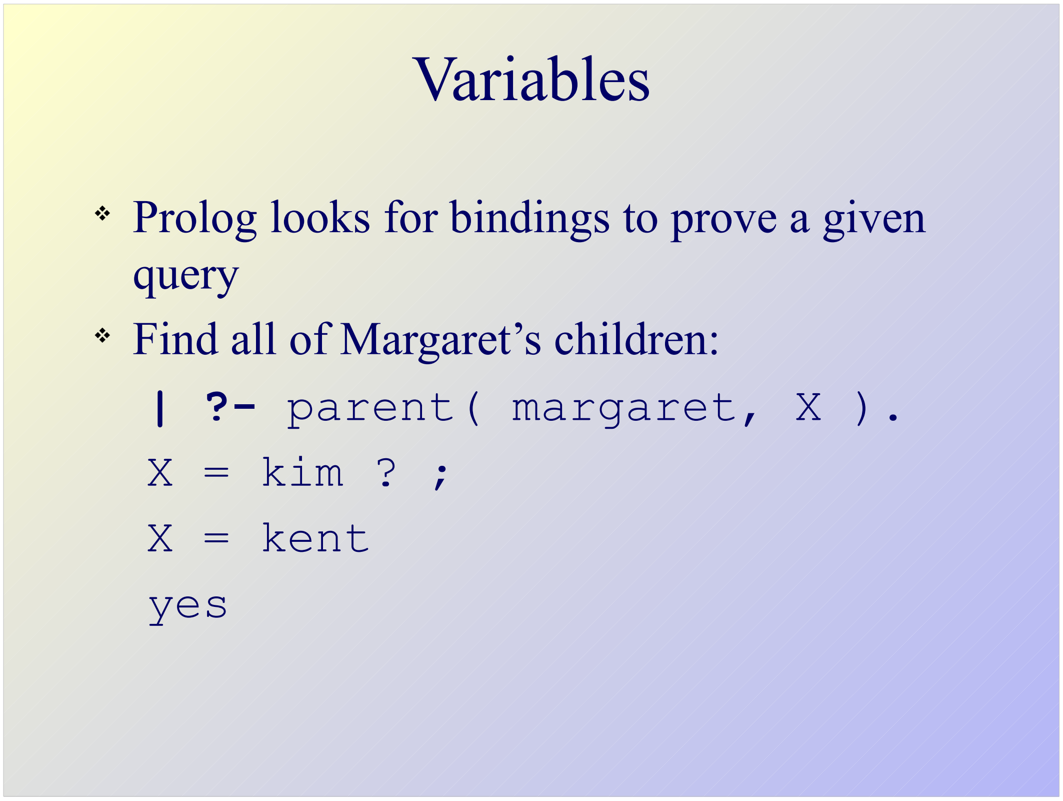
```
| ?- parent( esther, margaret ).
```

- Prolog looks through the DB, in order, to find a fact that matches the query. Found, so

```
yes
```

- If it fails to find one:

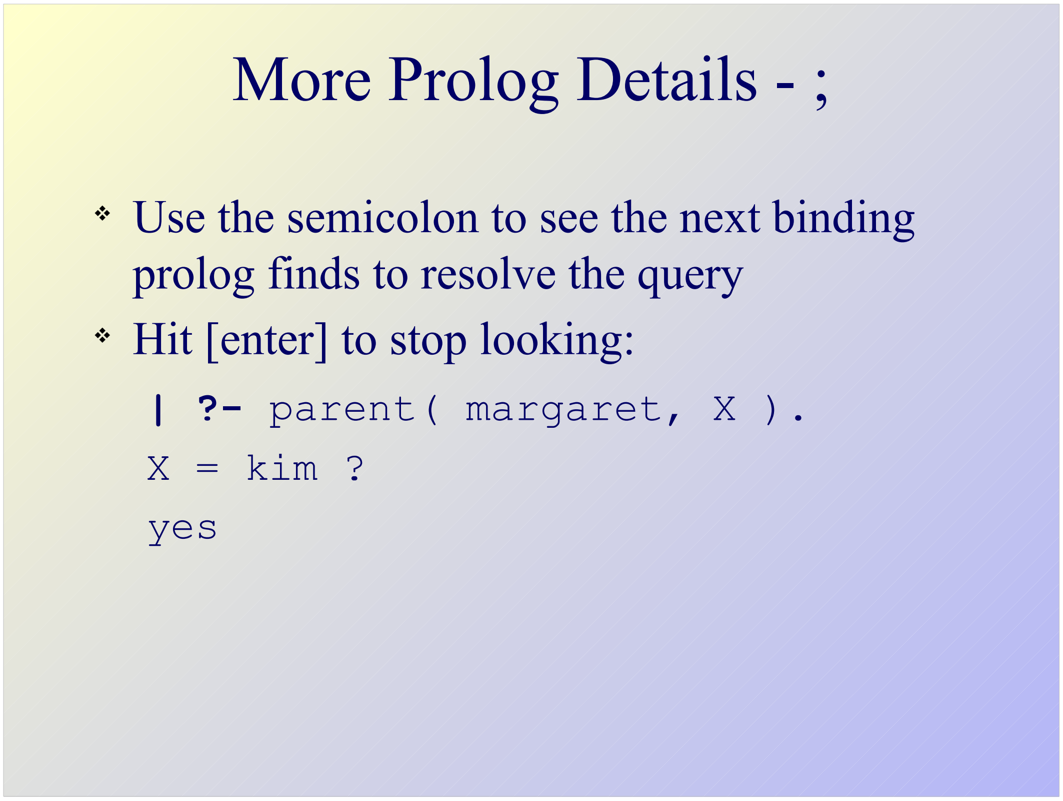
```
| ?- parent( esther, snoopy ).
no
```



Variables

- Prolog looks for bindings to prove a given query
- Find all of Margaret’s children:

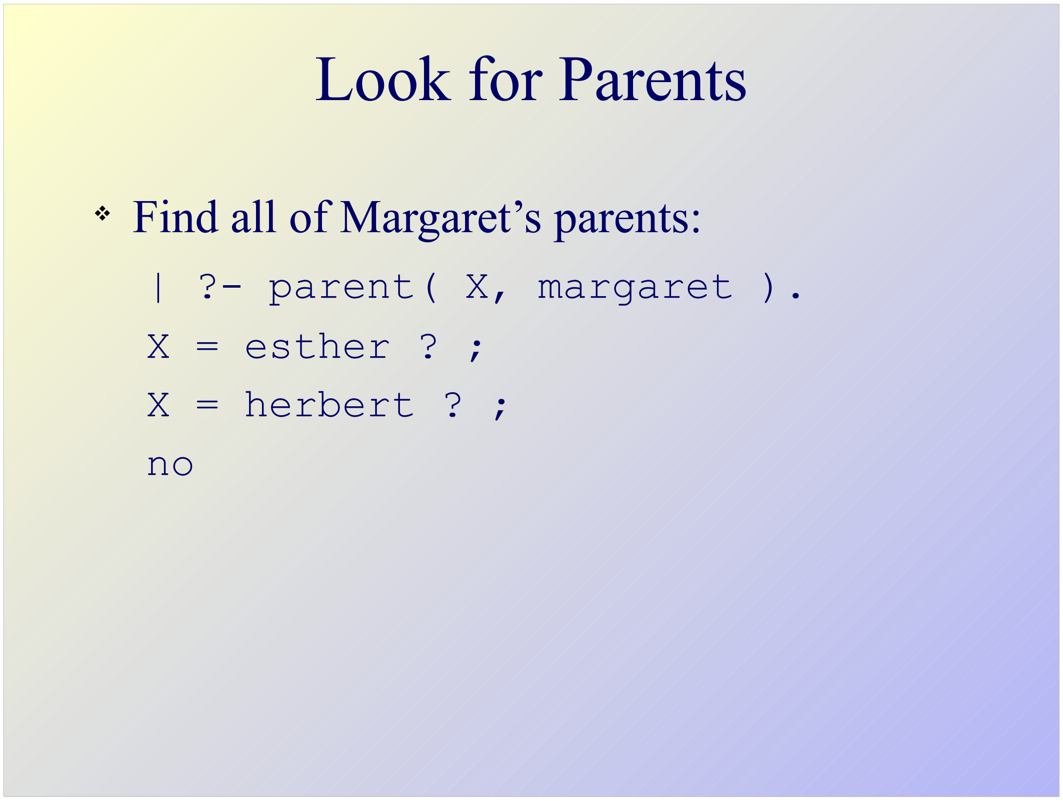
```
| ?- parent( margaret, X ).
X = kim ? ;
X = kent
yes
```



More Prolog Details - ;

- Use the semicolon to see the next binding prolog finds to resolve the query
- Hit [enter] to stop looking:

```
| ?- parent( margaret, X ).
X = kim ?
yes
```



Look for Parents

- Find all of Margaret’s parents:

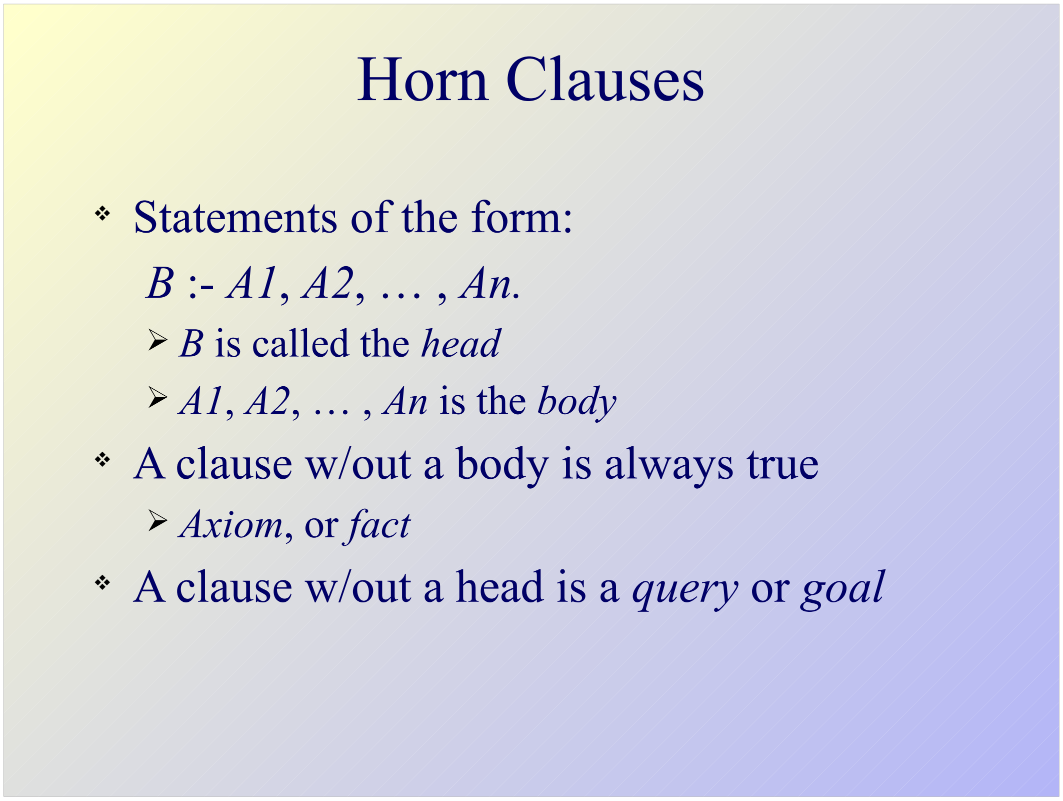
```
| ?- parent( X, margaret ).
X = esther ? ;
X = herbert ? ;
no
```

Conjunction of Terms

- Query can be a list of terms
  - Treated as a logical conjunction of the terms
- Find all who have both Esther and Herbert as parents:

```
   | ?- parent( esther, X ), parent( herbert,
X ).
   X = margaret ? ;
   no
```

  - Note, X is the same variable in both terms
    - "_" is special; a wildcard. Each instance is bound independently



Horn Clauses

- Statements of the form:

  B :- *A1*, *A2*, … , *An.*

  - *B* is called the *head*
  - *A1*, *A2*, … , *An* is the *body*
- A clause w/out a body is always true
  - *Axiom*, or *fact*
- A clause w/out a head is a *query* or *goal*

More Conjunction

- Find all of Kim’s grandparents:

```
| ?- parent( G, P ), parent( P, kim ).

G = esther
P = margaret ? ;

G = herbert
P = margaret ? ;

G = tom
P = dave

(15 ms) yes
```

Rules

- We can create more interesting *rules*, to prove more complex relations:

```
  | ?- [user].
  compiling user for byte code...
  ancestor( A, D ) :- parent( A, X ),
ancestor( X, D ).
  ancestor( A, A ).
```

GCD

```
gcd(U,0,U).

gcd(U,V,W) :- V=\=0, R is U mod V,
gcd(V,R,W).
```

```
  | ?- gcd( 15, 6, X ).
  X = 3 ?
  | ?- gcd( X, 6, 3 ).
  uncaught exception:
error(instantiation_error,(is)/2)
```

Lists

- [] – empty list
- [1, 2, 3] – finite list
- [X | Y]
  - X is the first element
  - Y is the rest

Examples

- member.pl
- lookup.pl
- append.pl
- sort.pl