

CS 152: *Programming Language Paradigms*



Introduction to Prolog

Prof. Tom Austin

San José State University

Deduction

Propositions:

- 1. Socrates is a man.*
- 2. All men are mortal.*

Conclusion:

Socrates is mortal

About Prolog

- **Programming in Logic**
 - Logic: "The science of reasoning and proof"
- *A declarative* programming language
 - you specify what you want
 - the computer determines how to do it
- *A logical* programming language
 - Relies on deductive reasoning
 - Reach conclusion from premises

References for Prolog

- "Learn Prolog Now",
<http://www.learnprolognow.org>
- SWI-Prolog website (contains manual and tutorials), <http://www.swi-prolog.org>
- "NLP with Prolog in the IBM Watson System",
<http://www.cs.nmsu.edu/ALP/2011/03/natural-language-processing-with-prolog-in-the-ibm-watson-system/>

Facts

Socrates is a man. Helen is a woman.

In Prolog:

man (socrates) .

woman (helen) .

Rules

Man is mortal. Woman is mortal.

In Prolog:


```
mortal(X) :- man(X) .
```

```
mortal(X) :- woman(X) .
```

X is a
variable



True if *either*
statement matches.



More facts and rules.

married(socrates) .

married(helen) .

husband(Person) :-

married(Person) ,

man(Person) .

comma
is "and"

Using not

```
immortal(zeus).
```

```
man(zeus).
```

```
mortal(X) :- man(X),  
             not(immortal(X)).
```

Alternate syntax for not

```
immortal(zeus).
```

```
man(zeus).
```

```
mortal(X) :- man(X),  
             \+immortal(X).
```

man(socrates) .

man(zeus) .

woman(helen) .

immortal(zeus) .

mortal(X) :- man(X), not(immortal(X)) .

mortal(X) :- woman(X) .

married(socrates) .

married(helen) .

husband(Person) :- married(Person),
man(Person) .

Socrates.prolog

Our knowledge base.

Loading Prolog file

```
$ swipl
```

```
Welcome to SWI-Prolog
```

```
...
```

```
?-
```

Loading Prolog file

```
$ swipl
```

```
Welcome to SWI-Prolog
```

```
...
```

```
?- [socrates].
```

```
true.
```

```
?-
```

Query: Is Socrates mortal?

```
$ swipl
```

```
Welcome to SWI-Prolog
```

```
...
```

```
?- [socrates].
```

```
true.
```

```
?- mortal(socrates).
```

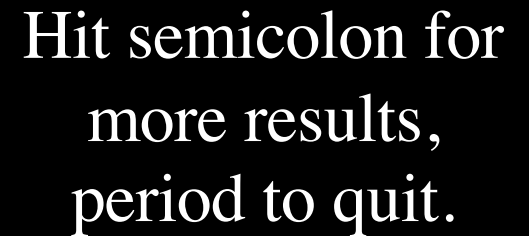
```
true.
```

Query: Who is mortal?

```
?- mortal(Person) .
```

```
Person = socrates ;
```

```
Person = helen .
```



Hit semicolon for
more results,
period to quit.

In class:

Game of Thrones in Prolog

Review: Facts

likes(batman, gotham).

likes(batman, justice).

likes(ras_al_ghul, justice).

likes(ras_al_ghul, revenge).

Review: Queries & Variables

What do Batman and Ra's al Ghul both like?

X is a
variable

comma
is "and"

```
?- likes (batman, X),  
likes (ras_al_ghul, X).
```

How does Prolog resolve queries?

Through 2 processes:

- *Resolution*
- *Unification*

Resolution & Unification

- **Resolution:** The process of matching facts & rules to perform *inferencing*
 - infer: derive logical conclusions from the rules.
 - If a subgoal matches the head of another rule, we can replace it with the body of the matching rule.
- **Unification:** Instantiation of variables via pattern matching

Query: likes(batman, X),
likes(ras_al_ghul, X).

Knowledge Base:

likes(batman, gotham).
likes(batman, justice).
likes(ras_al_ghul, justice).
likes(ras_al_ghul, revenge).

Query: `likes(batman, X),`
`likes(ras_al_ghul, X).`

Finds match for
first sub-query;
sets a *marker*

Knowledge Base:

`likes(batman, gotham).`
`likes(batman, justice).`
`likes(ras_al_ghul, justice).`
`likes(ras_al_ghul, revenge).`

Query: `likes(batman, gotham),`
`likes(ras_al_ghul,`
`gotham).`

Knowledge Base:

`likes(batman, gotham).`
`likes(batman, justice).`
`likes(ras_al_ghul, justice).`
`likes(ras_al_ghul, revenge).`

X is bound
to gotham

Query: likes (batman, gotham),
~~likes (ras_al_ghul,~~
~~gotham).~~

No match found:
fails and backtracks
to marker

ase:

likes (batman, gotham).
likes (batman, justice).
likes (ras_al_ghul, justice).
likes (ras_al_ghul, revenge).

Query: `likes(batman, X),`
`likes(ras_al_ghul, X).`

Knowledge Base:

`likes(batman, gotham).`

`likes(batman, justice).`

`likes(ras_al_ghul, justice).`

`likes(ras_al_ghul, revenge).`

Finds another
match for first
sub-query

Query: `likes(batman, justice),`
`likes(ras_al_ghul,`
`justice).`

Knowledge Base:

`likes(batman, gotham).`

`likes(batman, justice).`

`likes(ras_al_ghul, justice).`

`likes(ras_al_ghul, revenge).`

X is bound to
justice

Query: likes(batman, justice),
likes(ras_al_ghul,
justice).

Knowledge Base:

likes(batman, gotham).

likes(batman, justice).

likes(ras_al_ghul, justice).

likes(ras_al_ghul, revenge).

Match found,
and the result
is returned

**More
facts:**

```
villain(joker) .
```

```
villain(penguin) .
```

```
villain(catwoman) .
```

```
villain(scarecrow) .
```

```
kills_people(joker) .
```

```
kills_people(penguin) .
```

```
power(scarecrow, fear) .
```

```
romantic_interest(catwoman) .
```

```
romantic_interest(talia) .
```

Rules

Queries

```
scary(V) :- villain(V),  
            kills_people(V) .
```

"Head" of
the rule

```
scary(V) :- villain(V),  
            power(V, _) .
```

Who is scary?
(in-class)

Murder Mystery Lab

See Canvas for details.