

Prolog and first-order logic

Note Title

Propositional logic (PL) deals with variables that are true or false, and are combined using \wedge , \vee , \neg , \Rightarrow etc.

$$\text{e.g. } (P \vee Q) \wedge (Q \Rightarrow R)$$

First-order logic (FOL) (also known as predicate logic, predicate calculus etc)

extends PL with:

(1) objects e.g. cole, joanne, progLang

(2) relations e.g. student(cole)
takingCourse(joanne, progLang)

(3) quantifiers \forall (for all)
 \exists (there exists)

(4) variables e.g. X, Y

(5) other stuff we don't need in this course.

Notes:

1. We follow the Prolog convention of objects and relations start with lower case (e.g. cole) and variables with upper case (e.g. Who)

2. For a unary relation, $p(q)$, conventional reading is 'q is a p' or 'q has the property p'

e.g. $\text{red}(\text{apple})$ means "apple is red".

3. For binary relation, $p(q, r)$, conventional reading is " q has property p with respect to r "

e.g. $\text{older}(\text{cole}, \text{joanne})$ means
"cole is older than joanne"

Examples

$$\forall X \text{ csMajor}(X) \Rightarrow \text{student}(X)$$

$$\exists Y \text{ takingCourse}(Y, \text{progLang}) \wedge \text{mathMajor}(Y)$$

$$\exists Y \forall X \text{ takingCourse}(X, Y) \Rightarrow \text{philosophyMajor}(X)$$

Exercise: (1) translate each of the above into ordinary English.

(2) translate into FOL: All CS majors who are seniors are taking the senior seminar.

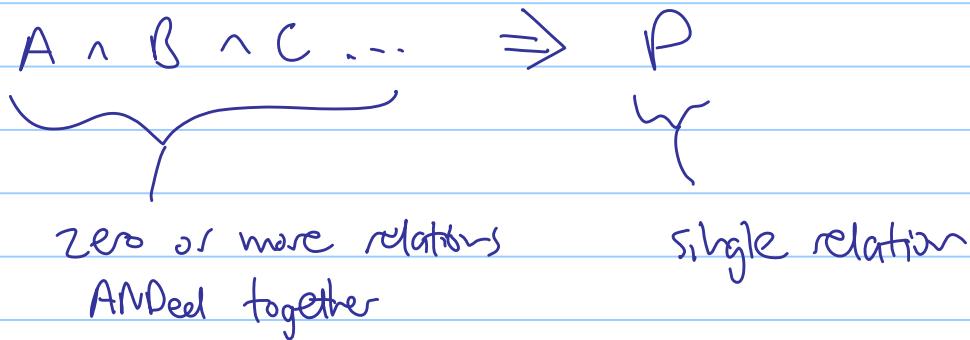
Why is this useful? FOL provides a rigorous framework for proving things. Given a knowledge base (KB) - a set of FOL statements known to be true - and a query, we can (often) infer automatically if the query is true.

That is, is the query entailed by the KB.

- This is exactly what a logic programming language like Prolog does for us. It first consults the KB, then tells us if it can succeed in proving the query true (i.e. is it entailed).
- One widely-used algorithm for inferring entailment is resolution. We don't study how it works, but you need to know:
 - if query Q is entailed by KB , the general resolution algorithm is guaranteed to prove it - but it could take a long time (exponential in size of inputs).
 - if Q is not entailed by KB , resolution might not terminate. (In fact, this problem is undecidable, so no algorithm can solve it).
 - Prolog implements a limited form of resolution. It's usually efficient, but it is not guaranteed to prove Q , even when Q is entailed!
 - Prolog works only on a limited form of KB: every statement in the KB must be a Horn clause. (see below).

Horn Clauses

A Horn clause looks like



However, because of Prolog's syntax, we will write all our Horn clauses backwards:

$$P \Leftarrow A \wedge B \wedge C .$$

examples:

$$\forall X \text{ takingCourse}(X, \text{senSem}) \Leftarrow \text{senior}(X) \wedge \text{csMajor}(X)$$

$$\text{student}(\text{phlc})$$

$$\forall X \text{ student}(X) \Leftarrow \text{senior}(X)$$

$$\forall X, Y, Z, \text{ teacherStudent}(X, Y) \Leftarrow \text{teachesCourse}(X, Z) \wedge \text{takingCourse}(Y, Z)$$

Horn clauses in Prolog

A Prolog program is just a list of Horn clauses, with a few notational conventions:

- ' (\Leftarrow) ' is ' $(:-)$ '
- ' (\wedge) ' is ' $(,)$ '
- All variables in a rule have ' \forall ' applied to them
(i.e. they are universally quantified)
- All variables in a query have ' \exists ' applied to them
(i.e. they are existentially quantified)
- All rules and facts are implicitly ANDed to obtain the KB.

Example:

Prolog

```
instructor (maccormick).  
instructor (X) :- teachesCourse (X, Y).  
takingCourse (senSem) :- senior (X), csMajor (X).
```

means in FOL:

```
 $\wedge \forall X, Y \text{ instructor}(X) \Leftarrow \text{teachesCourse}(X, Y)$   
 $\wedge \forall X \text{ takingCourse}(\text{senSem}) \Leftarrow \text{senior}(X) \wedge \text{csMajor}(X)$ 
```