Lecture #5: Logic Programming and Nested Quantifiers
Today’s topics

- Applications of predicate logic
- Nested quantifiers
Logic programming enables automated reasoning.

**Prolog**
- Programming in logic
- Developed in the 1970s for AI purposes

**Datalog**
- Logical formalization of databases
- Developed in the 1980s

For our purposes, we can consider Prolog and Datalog to be the same, though in reality they have very important differences.

Two main constructs:
- Facts
  - instructor(bill, cs441)
  - student(smith, cs441)
- Rules
  - teaches(P,S) :- instructor(P,C), student(S,C)

*Lower case = constant*

*Upper case = variable*
Facts define predicates by explicitly listing elements that satisfy those predicates

- “Dr. Garrison is the instructor for CS441”
  \[\text{instructor(bill, cs441)}\]

Rules define predicates by combining previously specified predicates

- “Professors teach the students enrolled in the courses for which they are the instructor”
  \[\text{teaches(P,S)} : - \text{instructor(P,C)}, \text{student(S,C)}\]

Prolog is an environment that lets us issue queries to determine which predicates are true!
A Security Example

grant(U, projector) :- located(U, 105), role(U, presenter)
located(U, R) :- owns(U, D), dev_loc(D, R)

role(bob, presenter)  owns(alice, laptop12)  dev_loc(laptop12, 105)
role(carol, presenter) owns(bob, tablet23)  dev_loc(tablet23, 105)
owns(carol, cell42)  dev_loc(cell42, 105)

Can Bob run the projector?

- Query: ?grant(bob, projector)
- Solution: true

Who is in room 105?

- Query: ?location(X, 105)
- Solution: alice, bob, carol
Write and evaluate the following queries

- Can Alice use the projector?
  - `?grant(alice, projector)`
  - `false`

- Can Carol use the projector
  - `?grant(carol, projector)`
  - `true`

- Which devices does Alice own?
  - `?owns(alice, X)`
  - `laptop12`
Logic programming is a useful tool!

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>19</td>
<td>555-1234</td>
</tr>
<tr>
<td>Danielle</td>
<td>33</td>
<td>555-5353</td>
</tr>
<tr>
<td>Zach</td>
<td>27</td>
<td>555-3217</td>
</tr>
<tr>
<td>Charlie</td>
<td>21</td>
<td>555-2335</td>
</tr>
</tbody>
</table>

Artificial Intelligence

Databases

Route planning

Security
Just for grins...

If you are interested in playing around with logic programming, download SWI-Prolog


This (free) package is a runtime environment in which you can write logic programs and evaluate queries.
Many times, we need the ability to **nest** one quantifier within the **scope** of another quantifier.

**Example:** All integers have an additive inverse. That is, for any integer $x$, we can choose an integer $y$ such that the sum of $x$ and $y$ is zero.

There is **no way** to express this statement using only a single quantifier!
Deciphering nested quantifiers isn’t as scary as it looks...

... you just read from left to right!

\[ \forall x \ \exists y \ \forall z \ [ (x + y) \times z = 0 ] \]

For all x...

... there exists a y such that...

... for all z...

... \((x + y) \times z = 0\)
A few more examples...

For all integers $x$ and for all integers $y$, $x + y = y + x$

For all integers $x$, for all integers $y$, and for all integers $z$, $(x+y)+z = x+(y+z)$

There exists an $x$ such that for all $y$, $x \times y = 0$
Since we always read from left to right, the order of quantifiers matters!

*Consider:* \( \forall x \ \exists y \ (x + y = 0) \)

Clearlly true! Just set \( y = -x \)

*Transpose:* \( \exists y \ \forall x \ (x + y = 0) \)

Not true...

*Remember:* As long as you read from left to right, you won’t have any problems!
Many mathematical statements can be translated into logical statements with nested quantifiers.

Translating mathematical expressions is often easier than translating English statements!

Steps:

1. Rewrite statement to make quantification and logical operators more explicit
2. Determine the order of which quantifiers should appear
3. Generate logical expression
Let’s try a translation...

**Statement:** Every real number except zero has a multiplicative inverse

\[ x \times y = 1 \]

**Rewrite:** For every real number \( x \), if \( x \neq 0 \), then there exists a real number \( y \) such that \( x \times y = 1 \).

\[ \forall x \quad (x \neq 0) \rightarrow \exists y \quad (x \times y = 1) \]
More examples...

**Statement:** The product of any two negative integers is always positive

**Statement:** For any real number $a$, it is possible to choose real numbers $b$ and $c$ such that $a^2 + b^2 = c^2$
Translating quantified statements to English is as easy as reading a sentence!

**Let:**
- $\text{C}(x) \equiv x$ is enrolled in CS441
- $\text{M}(x) \equiv x$ has an MP3 player
- $\text{F}(x, y) \equiv x$ and $y$ are friends
- Domain of $x$ and $y$ is “all students”

**Statement:** $\forall x \ [\text{C}(x) \rightarrow \text{M}(x) \lor (\exists y \ (\text{F}(x, y) \land \text{M}(y)))]$

For every student $x$...

... if $x$ is enrolled in CS441, then...

... $x$ has an MP3 player...

... or there exists another student $y$ such that...

... $x$ and $y$ are friends...

... and $y$ has an MP3 player.
Translate the following expressions into English

Let:

- O(x,y) ≡ x is older than y
- F(x,y) ≡ x and y are friends
- The domain for variables x and y is “all students”

**Statement:** \( \exists x \ \forall y \ O(x,y) \)

**Statement:** \( \exists x \ \exists y \ [F(x,y) \land \ \forall z \ [(y \neq z) \rightarrow \neg F(x,z)]] \)
Problem 1: Translate the following mathematical statement into predicate logic: Every even number is a multiple of 2. Assume that the predicate E(x) means “x is even.”

- **Hint:** What does “x is a multiple of 2” mean algebraically?

Problem 2: Translate the following expressions into English. Assume that C(x) means “x has a car”, F(x,y) means “x and y are friends”, and S(x) means “x is a student.”

- $\forall x (S(x) \rightarrow C(x)) \lor \exists y [F(x,y) \land C(y)]$
- $\forall x \exists y \exists z [C(x) \lor (F(x,y) \land C(y)) \lor (F(x,y) \land F(y,z) \land C(z))]$
Translating from English to a logical expression with nested quantifiers is a little bit more work...

**Steps:**

1. If necessary, rewrite the sentence to make quantifiers and logical operations more explicit
2. Create propositional functions to express the concepts in the sentence
3. State the domains of the variables in each propositional function
4. Determine the order of quantifiers
5. Generate logical expression
Let’s try an example…

Statement: Every student has asked at least one professor a question.

Rewrite: For every person x, if x is a student, then there exists a professor whom x has asked a question.

Let:
- $S(x) \equiv x$ is a student
- $P(x) \equiv x$ is a professor
- $Q(x,y) \equiv x$ has asked y a question

Translation: $\forall x (S(x) \rightarrow \exists y [P(y) \land Q(x,y)])$
Statement: There is a man who has tasted every type of beer.

Let:

- \( l_{M(x)} \equiv x \text{ is a man} \)
- \( l_{B(x)} \equiv x \text{ is a beer} \)
- \( l_{T(x,y)} \equiv x \text{ has tasted } y \)

Translation:

\[ \exists x \left( l_{M(x)} \land \forall y \left[ l_{B(y)} \rightarrow l_{T(x,y)} \right] \right) \]

Domain: all people

Domain: all drinks

Domains: \( x = \text{ all people} \), \( y = \text{ all drinks} \)
Negating expression with nested quantifiers is actually pretty easy...

... you just *repeatedly* apply DeMorgan’s laws!

\[
\neg [ \exists x (M(x) \land \forall y [B(y) \rightarrow T(x,y)])]
\]

\[
a \rightarrow b \equiv \neg a \lor b
\]

*In English:* For all people \( x \), if \( x \) is a man, then there exists some type beer that \( x \) has not tasted.

*Alternatively:* No man has tasted every type of beer.
A few stumbling blocks...

Whether the negation sign is on the inside or the outside of a quantified statement makes a big difference!

**Example:** Let $T(x) \equiv “x$ is tall”. Consider the following:

- $\neg \forall x \ T(x)$
  - “It is not the case that all people are tall.”

- $\forall x \ \neg T(x)$
  - “For all people $x$, it is not the case that $x$ is tall.”

**Note:** $\neg \forall x \ T(x) = \exists x \ \neg T(x) \neq \forall x \ \neg T(x)$

**Recall:** When we push negation into a quantifier, DeMorgan’s law says that we need to **switch** the quantifier!
Let:  

\( C(x) \equiv \text{“x is enrolled in CS441”} \)  
\( S(x) \equiv \text{“x is smart.”} \)

Question: The following two statements look the same, what’s the difference?

- \( \exists x \ [C(x) \land S(x)] \)  
- \( \exists x \ [C(x) \rightarrow S(x)] \)

Subtle note: The second statement is true if there exists one person not in CS441, because F→F or F→T.

There is a smart student in CS441.

There exists a student \( x \) such that if \( x \) is in CS441, then \( x \) is smart.
Negate \( \forall x \ (S(x) \rightarrow \exists y \ [P(y) \land Q(x,y)]) \)

\( \neg \forall x \ (S(x) \rightarrow \exists y \ [P(y) \land Q(x,y)]) \)

In English: There exists a student \( x \) such that for all people \( y \), if \( y \) is a professor then \( x \) has not asked \( y \) a question.

Alternatively: There exists a student that has never asked any professor a question.
Problem 3: Translate the following English sentences into predicate logic.

a) Every student has at least one friend that is dating Penguins fan.

b) If a person is a parent and a man, then they are the father of some child.

Problem 4: Negate the results from Problem 3 and translate the negated expressions back into English.
Logic programming is an interesting application of predicate logic that is used throughout computer science.

Quantifiers can be nested
- Nested quantifiers are read left to right
- Order is important!
- Translation and negation work the same as they did before!

Next lecture:
- Rules of inference
- Please read sections 1.6 - 1.7