Predicate Calculus

Chapter 12, Section 3

Predicate calculus

- *Predicate*: function that maps constants and variables to `true` and `false`
- *First order predicate calculus*: notation and inference rules for constructing and reasoning about propositions:
- Operators:
  - and $\wedge$
  - or $\vee$
  - not $\neg$
  - implication $\rightarrow$
  - equivalence $\leftrightarrow$
- Quantifiers:
  - existential $\exists$
  - universal $\forall$

p	q	$p \vee q$	$p \wedge q$	$\neg p$	$p \rightarrow q$	$p \leftrightarrow q$
T	T	T	T	F	T	T
T	F	T	F	F	F	F
F	T	T	F	T	T	F
F	F	F	F	T	T	T

Predicate calculus

- Examples

city

$$\forall C(\texttt{rainy}(C) \wedge \texttt{cold}(C) \to \texttt{snowy}(C))$$

students.

class

$$\forall A, \forall B(\texttt{takes}(A, C) \wedge \texttt{takes}(B, C) \to \texttt{classmates}(A, B))$$

- Fermat's last Theorem:

$$\forall N\,((N > 2) \to \neg(\exists A\,\exists B\,\exists C(A^N + B^N = C^N)))$$

- ∀, ∃ bind variables like λ in λ-calculus

Predicate calculus

- Normal form
  - the same thing can be written in different ways:

$$(P \rightarrow Q) \equiv (\neg P \vee Q)$$

$$\neg \exists X\,(P(X)) \equiv \forall X\,(\neg P(X))$$

$$\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$$

  - This is good for humans, bad for machines
  - Automatic theorem proving requires a normal form

$\forall X(\neg\texttt{student}(X) \rightarrow (\neg\texttt{resident}(X) \wedge \neg\exists Y(\texttt{takes}(X, Y) \wedge \texttt{class}(Y))))$

1) remove $\rightarrow$, $\leftrightarrow$ using $p \rightarrow q \equiv \neg p \vee q$

$\Rightarrow \forall x S(x) \vee (\neg R(x) \wedge \neg \exists Y (T(x, Y) \wedge C(Y))$

2) move negation inward using De Morgan law.

$\Rightarrow \forall x S(x) \vee (\neg R(x) \wedge \forall Y (\neg T(x, Y) \vee \neg C(Y))$

3) pull universal quantifier to the front, drop the universial quantifiers.

$\Rightarrow \forall x \forall Y S(x) \vee (\neg R(x) \wedge (\neg T(x, Y) \vee \neg C(Y))$

$\Rightarrow S(x) \vee (\neg R(x) \wedge (\neg T(x, Y) \vee \neg C(Y))$

4) convert it to $\overset{\wedge}{\text{conjunctions}}$ of $\overset{\vee}{\text{disjunctions}}$

$\Rightarrow (S(x) \vee \neg R(x)) \wedge (\neg T(x, Y) \vee \neg C(Y) \vee S(x))$

$\Rightarrow (S(x) \rightarrow R(x)) \wedge (T(x, Y) \wedge C(Y) \rightarrow S(x))$

$\Rightarrow S(x) \leftarrow R(x) \wedge (S(x) \leftarrow T(x, Y) \wedge C(Y))$

Clausal Form

- *Clausal form*
- Example:

$\forall X\,(\neg\texttt{student}(X) \rightarrow (\neg\texttt{resident}(X) \wedge \neg\exists Y\,(\texttt{takes}(X, Y) \wedge \texttt{class}(Y))))$

- 1. eliminate $\rightarrow$ and $\leftrightarrow$:

$\forall X\,(\texttt{student}(X) \vee (\neg\texttt{resident}(X) \wedge \neg\exists Y\,(\texttt{takes}(X, Y) \wedge \texttt{class}(Y))))$

Clausal Form

$\forall X\,(\mathtt{student}(X) \lor (\neg\mathtt{resident}(X) \land \neg\exists Y\,(\mathtt{takes}(X, Y) \land \mathtt{class}(Y))))$

- 2. move ¬ inward (using De Morgan's laws):

$\forall X\,(\mathtt{student}(X) \lor (\neg\mathtt{resident}(X) \land \forall Y\,(\neg(\mathtt{takes}(X, Y) \land \mathtt{class}(Y)))))$

$\equiv$

$\forall X\,(\mathtt{student}(X) \lor (\neg\mathtt{resident}(X) \land \forall Y\,(\neg\mathtt{takes}(X, Y) \lor \neg\mathtt{class}(Y))))$

Clausal Form

$\forall X\,(\texttt{student}(X) \lor (\lnot\texttt{resident}(X) \land \forall Y\,(\lnot\texttt{takes}(X, Y) \lor \lnot\texttt{class}(Y))))$

- 3. eliminate existential quantifiers
  - Skolemization (not necessary in our example)
- 4. pull universal quantifiers to the outside of the proposition (some renaming might be needed)

$\forall X\,\forall Y\,(\texttt{student}(X) \lor (\lnot\texttt{resident}(X) \land (\lnot\texttt{takes}(X, Y) \lor \lnot\texttt{class}(Y))))$

  - convention: rules are universally quantified
    - we drop the implicit ∀'s:

$\texttt{student}(X) \lor (\lnot\texttt{resident}(X) \land (\lnot\texttt{takes}(X, Y) \lor \lnot\texttt{class}(Y)))$

Clausal Form

student(X) ∨ (¬resident(X) ∧ (¬takes(X, Y) ∨ ¬class(Y)))

- 5. convert the proposition in *conjunctive normal form* (*CNF*)
  - conjunction of disjunctions

(student(X) ∨ ¬resident(X)) ∧
(student(X) ∨ ¬takes(X, Y) ∨ ¬class(Y))

Clausal Form

$(\texttt{student}(X) \vee \neg\texttt{resident}(X)) \;\wedge$
$(\texttt{student}(X) \vee \neg\texttt{takes}(X, Y) \vee \neg\texttt{class}(Y))$

- We can rewrite as:

$(\texttt{resident}(X) \rightarrow \texttt{student}(X)) \;\wedge$
$((\texttt{takes}(X, Y) \wedge \texttt{class}(Y)) \rightarrow \texttt{student}(X))$

$\equiv$

$(\texttt{student}(X) \leftarrow \texttt{resident}(X)) \;\wedge$
$(\texttt{student}(X) \leftarrow (\texttt{takes}(X, Y) \wedge \texttt{class}(Y)))$

Clausal Form

- We obtained:

$(\texttt{student}(X) \leftarrow \texttt{resident}(X)) \wedge$
$(\texttt{student}(X) \leftarrow (\texttt{takes}(X, Y) \wedge \texttt{class}(Y)))$

- which translates directly to Prolog:

```
student(X) :- resident(X).
student(X) :- takes(X, Y), class(Y).
```

`:-` means "if"

`,` means "and"

Horn Clauses

- *Horn clauses*
  - particular case of clauses: only one non-negated term:

$$\neg Q_1 \vee \neg Q_2 \vee \ldots \vee \neg Q_k \vee P \equiv \neg(Q_1 \vee Q_2 \cdots Q_k) \vee P$$

$$Q_1 \wedge Q_2 \wedge \ldots \wedge Q_k \rightarrow P \equiv$$

$$P \leftarrow Q_1 \wedge Q_2 \wedge \ldots \wedge Q_k$$

  - which is a *rule* in Prolog:

```
P :- Q1, Q2,...,Qk.
```

  - for $k = 0$ we have a *fact*:

```
P.
```

Automated proving

- Rule: both sides of `:-`

`P :- Q1, Q2,...,Qk.` means $P \leftarrow Q_1 \wedge Q_2 \wedge \ldots \wedge Q_k$

- Fact: left-hand side of (implicit) `:-`

`P.` means $P \leftarrow \texttt{true}$

- Query: right-hand side of (implicit) `:-`

`?- Q1, Q2,...,Qk.`

- *Automated proving*: given a collection of axioms (facts and rules), add the *negation* of the theorem (query) we want to prove and attempt (using *resolution*) to obtain a contradiction
  - Query negation: $\neg(Q_1 \wedge Q_2 \wedge \ldots \wedge Q_k)$

Automated proving

- Example

```
student(john).
?- student(john).
true.
```

- Fact: $\texttt{student}(\text{john}) \leftarrow \texttt{true}$
- Query (negated):

$$\neg\texttt{student}(\text{john}) \equiv \texttt{false} \leftarrow \texttt{student}(\text{john})$$

- We obtain a contradiction (that proves the query):

$$\texttt{false} \leftarrow \texttt{student}(\text{john}) \leftarrow \texttt{true}$$

- The above contradiction is obvious; in general, use *resolution*.

Resolution

- *Resolution* (propositional logic):
  - From hypotheses:

$$(A_1 \lor A_2 \;\lor\ldots\lor A_k \lor C) \land (B_1 \lor B_2 \;\lor\ldots\lor B_l \lor \neg C)$$

  - We can obtain the conclusion:

if C is true, ...
false, ...

$$A_1 \lor A_2 \;\lor\ldots\lor A_k \lor B_1 \lor B_2 \;\lor\ldots\lor B_l$$

- Example: *modus ponens*

$p \to q \land p$ gives q (because $p \to q$ is $\neg p \lor q$)

- In predicate logic:
  - C and $\neg C'$: where C, C' may not be identical but can be *unified*: that means, they can be made identical by substituting variables (details later)

Resolution example

```
student(X) :- resident(X).
student(X) :- takes(X, Y), class(Y).
resident(john).
takes(mark, 3342).
class(3342).

?- student(john).
true
```

- Resolution (add negation of query):

$(\neg \texttt{resident}(X) \vee \texttt{student}(X)) \wedge$
$(\neg \texttt{takes}(Y, Z) \vee \neg \texttt{class}(Z) \vee \texttt{student}(Y)) \wedge$
$\texttt{resident}(\text{john}) \wedge$
$\texttt{takes}(\text{mark}, 3342) \wedge$
$\texttt{class}(3342) \wedge$
$\neg \texttt{student}(\text{john})$

Resolution example

$(\neg \texttt{resident}(X) \lor \texttt{student}(X)) \land$
$(\neg \texttt{takes}(Y, Z) \lor \neg \texttt{class}(Z) \lor \texttt{student}(Y)) \land$
$\texttt{resident}(\text{john}) \land$
$\texttt{takes}(\text{mark}, 3342) \land$
$\texttt{class}(3342) \land$
$\neg \texttt{student}(\text{john})$

- $\texttt{student}(X)$ and $\texttt{student}(\text{john})$ *unify* for $X = \text{john}$

$(\neg \texttt{resident}(\text{john}) \lor \texttt{student}(\text{john})) \land$
$(\neg \texttt{takes}(Y, Z) \lor \neg \texttt{class}(Z) \lor \texttt{student}(Y)) \land$
$\texttt{resident}(\text{john}) \land$
$\texttt{takes}(\text{mark}, 3342) \land$
$\texttt{class}(3342) \land$
$\neg \texttt{student}(\text{john})$

Resolution example

(¬resident(john) ∨ student(john)) ∧
(¬takes(Y, Z) ∨ ¬class(Z) ∨ student(Y)) ∧
resident(john) ∧
takes(mark, 3342) ∧
class(3342) ∧
¬student(john)

- resolution gives:

¬resident(john) ∧
(¬takes(Y, Z) ∨ ¬class(Z) ∨ student(Y)) ∧
resident(john) ∧
takes(mark, 3342) ∧
class(3342)

Resolution example

$\neg$resident(john) $\wedge$
($\neg$takes(Y, Z) $\vee$ $\neg$class(Z) $\vee$ student(Y)) $\wedge$
resident(john) $\wedge$
takes(mark, 3342) $\wedge$
class(3342)

- Resolution gives:

($\square$) $\wedge$ ← empty clause
($\neg$takes(Y, Z) $\vee$ $\neg$class(Z) $\vee$ student(Y)) $\wedge$
takes(mark, 3342) $\wedge$
class(3342)

- The empty clause ($\square$) is not satisfiable
- We obtained a contradiction showing that student(john) is *provable* from the given axioms

Resolution example

```
?- student(matthew).
false.
```

- Resolution:

matthew

$(\neg \texttt{resident}(X) \lor \texttt{student}(X)) \land$
$(\neg \texttt{takes}(Y, Z) \lor \neg \texttt{class}(Z) \lor \texttt{student}(Y)) \land$
$\texttt{resident}(\text{john}) \land$
$\texttt{takes}(\text{mark}, 3342) \land$
$\texttt{class}(3342) \land$
$\neg \texttt{student}(\text{matthew})$

$\neg \texttt{resident}(\text{matthew}) \land$
$(\neg \texttt{takes}(Y, Z) \lor \neg \texttt{class}(Z) \lor \texttt{student}(Y)) \land$
$\texttt{resident}(\text{john}) \land$
$\texttt{takes}(\text{mark}, 3342) \land$
$\texttt{class}(3342)$

Resolution example

$\neg$`resident`(matthew) ∧

($\neg$`takes`(Y, 3342) ∨ `student`(Y)) ∧

`resident`(john) ∧

`takes`(mark, 3342)

$\neg$`resident`(matthew) ∧

`student`(mark) ∧

`resident`(john)

- cannot obtain a contradiction
- `student`(matthew) is *not provable* from the given axioms

Skolemization

- So far we did not worry about existential quantifiers
- What if we have:

$$\exists X\,(\texttt{takes}(X, 3342) \wedge \texttt{year}(X, 2))$$

- To get rid of the ∃, we introduce a constant, a, (as a notation for the one which is assumed to exists by ∃)

$$\texttt{takes}(a, 3342) \wedge \texttt{year}(a, 2)$$

Skolemization

- What if we do this inside the scope of a universal quantifier ∀:

$$\forall X\,(\neg\mathtt{resident}(X) \vee \exists Y\,(\mathtt{address}(X, Y)))$$

- We get rid again of ∃ by choosing an address which depends on X, say $\mathtt{ad}(X)$:

$$\forall X\,(\neg\mathtt{resident}(X) \vee (\mathtt{address}(X, \mathtt{ad}(X))))$$

Skolemization

- In Prolog

```
takes(a, 3342).
year(a, 2).
address(X, ad(X)) :- resident(X).
class_with_2nd(C) :- takes(X, C), year(X, 2).
has_address(X) :- address(X, Y).
resident(b).


?- class_with_2nd(C).
C = 3342
?- has_address(X).
X = b
```

Skolemization

```
?- takes(X, 3342).
X = a
```

- We cannot identify a 2nd-year student in 3342 by name

```
?- address(b, X).
X = ad(b).
```

- We cannot find out the address of b

Horn Clauses Limitations

- Horn clauses: only *one* non-negated term (*head*):

$$\neg Q_1 \vee \neg Q_2 \ \vee \ldots \vee \neg Q_k \ \vee P \ \equiv \ P \leftarrow Q_1 \wedge Q_2 \ \wedge \ldots \wedge Q_k$$

```
P :- Q1, Q2,...,Qk.
```

- If we have *more than one* non-negated term (two heads):

$$\neg Q_1 \vee \neg Q_2 \ \vee \ldots \vee \neg Q_k \ \vee P_1 \vee P_2 \equiv P_1 \vee P_2 \leftarrow Q_1 \wedge Q_2 \ \wedge \ldots \wedge Q_k$$

- then we have a disjunction in the left-hand side of $\leftarrow$ (`:-`)

```
P1 or P2 :- Q1, Q2,...,Qk.
```

- which is not allowed in Prolog

Horn Clauses Limitations

- If we have *less than one* (zero) non-negated terms:

$$\neg Q_1 \vee \neg Q_2 \ \vee \ldots \vee \neg Q_k$$

$$\equiv$$

$$\texttt{false} \leftarrow Q_1 \wedge Q_2 \ \wedge \ldots \wedge Q_k$$

- the closest we have is:

```
:- Q1, Q2,...,Qk.
```

- which Prolog allows a query, not a rule

Horn Clauses Limitations

- Example: two heads

  "every living thing is an animal or a plant"
- Clausal form:

  $\texttt{animal}(X) \vee \texttt{plant}(X) \leftarrow \texttt{living}(X) \equiv$

  $\texttt{animal}(X) \vee \texttt{plant}(X) \vee \neg\texttt{living}(X)$
- In Prolog, the closest we can do is:

```
animal(X) :- living(X), not(plant(X)).
plant(X) :- living(X), not(animal(X)).
```

- which is not the same, because, as we'll see later, `not` indicates Prolog's inability to prove, not falsity