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 $\forall$

X —> Y => ~X v Y
0 —> Y is always true

Predicate calculus

- Examples

$$\forall \underline{C}(\texttt{rainy}(\underline{C}) \land \texttt{cold}(\underline{C}) \rightarrow \texttt{snowy}(\underline{C}))$$

all C bound here

$$\forall \underline{A},\forall \underline{B}(\texttt{takes}(\underline{A}, C) \land \texttt{takes}(\underline{B}, C) \rightarrow \texttt{classmates}(\underline{A}, \underline{B}))$$

A and B are bound here

but C is free

- Fermat's last Theorem:

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

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

Predicate calculus

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

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

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

$$\neg(P \land Q) \equiv (\neg P \lor \neg Q)$$

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

Clausal Form

we also called it normal form

- *Clausal form*
- Example:

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

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

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

Clausal Form

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

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

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

≡

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

Clausal Form

$\forall X$ (`student`(X) ∨ (¬`resident`(X) ∧ $\forall Y$ (¬`takes`(X, Y) ∨ ¬`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)

$\forall X\,\forall Y$ (`student`(X) ∨ (¬`resident`(X) ∧ (¬`takes`(X, Y) ∨ ¬`class`(Y))))

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

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

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

- We can rewrite as:

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

$\equiv$

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

两条prolog

Clausal Form

- We obtained:

$(\mathtt{student}(X) \leftarrow \mathtt{resident}(X)) \land$
$(\mathtt{student}(X) \leftarrow (\mathtt{takes}(X, Y) \land \mathtt{class}(Y)))$

- 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:

$$\neg Q_1 \lor \neg Q_2 \lor \ldots \lor \neg Q_k \lor P \equiv$$
$$Q_1 \land Q_2 \land \ldots \land Q_k \rightarrow P \equiv$$
$$P \leftarrow Q_1 \land Q_2 \land \ldots \land Q_k$$

如果有第二个non-negated term
这时候转化成 imply 就会导致
第二个non-negated term变成
negated的了

prolog 不存在这种形式

  - 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 $P \leftarrow$ `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: `student`(john) ← `true`
- Query (negated):

  ¬`student`(john) ≡ `false` ← `student`(john)

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

`false` ← `student`(john) ← `true`

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

Resolution

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

$(A_1 \vee A_2 \vee \ldots \vee A_k \vee C) \wedge (B_1 \vee B_2 \vee \ldots \vee B_l \vee \neg C)$

  - We can obtain the conclusion:

$A_1 \vee A_2 \vee \ldots \vee A_k \vee B_1 \vee B_2 \vee \ldots \vee B_l$

- Example: *modus ponens*

$p \rightarrow q \wedge p$ gives q (because $p \rightarrow q$ is $\neg p \vee q$)

- 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):

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

Resolution example

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

- student(X) and student(john) *unify* for X = john

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

($\square$) $\wedge$
($\neg$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:

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

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

Resolution example

¬resident(matthew) ∧

(¬takes(*Y*, 3342) ∨ student(*Y*)) ∧

resident(john) ∧

takes(mark, 3342)

¬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:

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

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

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

Skolemization

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

$$\forall X\,(\neg \texttt{resident}(X) \lor \exists Y\,(\texttt{address}(X, Y)))$$

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

$$\forall X\,(\neg \texttt{resident}(X) \lor (\texttt{address}(X, \texttt{ad}(X))))$$

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:

$$\neg Q_1 \lor \neg Q_2 \lor \ldots \lor \neg Q_k$$

$$\equiv$$

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

- 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:

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

  $\texttt{animal}(X) \lor \texttt{plant}(X) \lor \neg\texttt{living}(X)$
- 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