EECS 3401 — AI and Logic Prog. — Lecture 6

Adapted from slides of Prof. Yves Lesperance

Vitaliy Batusov
vbatusov@cse.yorku.ca

York University

September 30, 2020

- Today: **Inference in First-Order Logic**
- Required reading: Russell & Norvig, Chapters 9.1, 9.2, 9.5

Computing Logical Consequences

Recap:

- KB is a set of axioms
- Reasoning $=$ finding logical consequences of a KB
- Logical consequence is a semantic notion
  If a formula ϕ holds in every model of a KB, then KB $\models \phi$

- Can this be done mechanically/in code?
- Yes. There are procedures for generating logical consequences
- Called **proof procedures**

Proof Procedures

- Proof procedures operate by simply manipulating formulas. They pay no heed whatsoever to interpretations
- Still, *they respect the semantics of the interpretations!*
- We will develop a proof procedure for FOL called **resolution**
  Prolog uses resolution, you've seen it work already.

Properties of Proof Procedures

Notation: $KB \vdash \phi$ means that formula ϕ *can be proved* from the KB (using some implicit proof procedure).

- Generally speaking, what properties do we expect a proof procedure to have?

Properties of Proof Procedures

Two fundamental properties:

Soundness If $KB \vdash \phi$, then $KB \models \phi$
(if we derive a formula from KB via the proof procedure, it better be a logical consequence of KB)

Completeness If $KB \models \phi$, then $KB \vdash \phi$
(if a formula is a logical consequence, our proof procedure should be capable of deriving it from KB)

Note: proof procedures are computable, but they might have very bad complexity in the worst case. Completeness is not necessarily achievable in practice

Resolution: Clausal Form

- Resolution works with formulas in a particular form — **clausal form**
- A **literal** is an atomic formula or the negation of an atomic formula
  A literal: $dog(fido)$. Also a literal: $\neg cat(fido)$.
  Not a literal: $cat(fido) \vee dog(fido)$ because contains disjunction
- A **clause** is a *disjunction of literals*
  A clause: $cat(fido) \vee dog(fido)$
  Also a clause: $\neg owns(fido, fred) \vee \neg dog(fido) \vee person(fred)$
  Not a clause: $\neg cat(fido) \wedge \neg dog(fido)$ because contains conjunction
- Since a clause is always a disjunction, we can treat it as a collection/tuple of literals:

$$(\neg owns(fido, fred), \neg dog(fido), person(fred))$$

- A **clausal theory** is a conjunction of clauses

Clausal Form

- A **Horn clause** is a clause with no more than one positive literal

  A Horn clause: $(\neg owns(fido, fred), \neg dog(fido), person(fred))$

  Not a Horn clause: $(cat(fido), dog(fido))$

- *Prolog programs are clausal theories. Every fact or rule in a Prolog program is a Horn clause.*

$\neg q_1 \vee \neg q_2 \vee \ldots \vee \neg q_n \vee p$	Horn clause
$(q_1 \wedge q_2 \wedge \ldots \wedge q_n) \rightarrow p$	same, as an implication
`p :- q_1, q_2, ... , q_n.`	same, as a Prolog rule

Resolution Rule for Ground Clauses

Basic Principle of Resolution: From two clauses

$$(p, q_1, q_2, \ldots, q_k) \quad \text{and} \quad (\neg p, r_1, r_2, \ldots, r_n),$$

infer the new clause

$$(q_1, q_2, \ldots, q_k, r_1, r_2, \ldots, r_n).$$

Example: from

$$(\neg largerThan(clyde, cup), \neg fitsIn(clyde, cup)) \quad \text{and} \quad (fitsIn(clyde, cup)),$$

infer the new clause

$$(\neg largerThan(clyde, cup)).$$

Resolution Proof: Forward Chaining

Logical consequences can be generated from the resolution rule in two ways: **Forward Chaining** and **Refutation**.

Forward Chaining Inference: chain multiple resolution steps

- Suppose we have a sequence of clauses $C_1, C_2, \ldots, C_k$
- Suppose that each C_i is either in the KB or is the result of a resolution step involving two prior clauses in the sequence
- Then, we have that $KB \vdash C_k$

Forward chaining is sound, so we also have $KB \models C_k$

Resolution Proof: Refutation Proofs

Refutation Proofs: a proof by contradiction

- Fact: $KB \models \phi$ iff $KB \land \neg\phi$ has no model ("unsatisfiable")
- Since resolution is sound, then if we can derive a **contradiction** from $KB \land \neg\phi$, we consider ϕ proved
- A **contradiction** is the **empty clause** ()

A proof:

- Suppose we have a sequence of clauses $C_1, C_2, \ldots, C_m$
- Suppose that each C_i is either in the $KB \land \neg\phi$ or is the result of resolving two prior clauses in the sequence
- Suppose C_m is ()
- Then $KB \vdash \phi$

By soundness, $KB \models \phi$.

Resolution Proof Example: Forward Chaining

Knowledge Base: (in clausal form)[1]

$$(elephant(clyde), giraffe(clyde)) \quad (1)$$
$$(\neg elephant(clyde), likes(clyde, peanuts)) \quad (2)$$
$$(\neg giraffe(clyde), likes(clyde, leaves)) \quad (3)$$
$$(\neg likes(clyde, leaves)) \quad (4)$$

Want to prove: $likes(clyde, peanuts)$

Using forward chaining:

- Resolve (3) & (4), get new clause (5) = $(\neg giraffe(clyde))$
- Resole (5) & (1), get new clause (6) = $(elephant(clyde))$
- Resolve (6) & (2), get desired conclusion $(likes(clyde, peanuts))$

[1] Reminder: $A \rightarrow B$ stands for $\neg A \vee B$

Resolution Proof Example: Refutation

Knowledge Base:

$$(elephant(clyde), giraffe(clyde)) \tag{1}$$
$$(\neg elephant(clyde), likes(clyde, peanuts)) \tag{2}$$
$$(\neg giraffe(clyde), likes(clyde, leaves)) \tag{3}$$
$$(\neg likes(clyde, leaves)) \tag{4}$$

Want to prove: $likes(clyde, peanuts)$

Using refutation:

- Add negation of query to KB: (5) $= \neg likes(clyde, peanuts)$
- Resolve (5) & (2), get (6) $= (\neg elephant(clyde))$
- Resolve (6) & (1), get (7) $= (giraffe(clyde))$
- Resolve (7) & (3), get (8) $= (likes(clyde, leaves))$
- Resolve (8) & (4), get **the empty clause** ()

Resolution Proofs

- Proofs by refutation have the advantage that they are easier to find — they are more focused on the particular conclusion we are trying to reach

- To develop a complete resolution proof procedure for First-Order Logic, we need:
  1. A way of converting a KB and the query ϕ into clausal form [Today's focus]
  2. A way of extending resolution to work on formulas with variables [unification! Will cover next Monday]

Conversion to Clausal Form

An 8-step procedure to convert a KB into clausal form:

1. Eliminate implications
2. Move negations inwards and simplify $\neg\neg q$
3. Standardize variables
4. Skolemize
5. Convert to Prenex form
6. Distribute conjunctions over disjunctions
7. Flatten nested conjunctions and disjunctions
8. Convert to clauses

Will use this example to illustrate:

$$\forall X \left\{ p(X) \rightarrow \left[\forall Y \left[p(Y) \rightarrow p(f(X, Y)) \right] \wedge \neg \left[\forall Y (\neg q(X, Y) \wedge p(Y)) \right] \right] \right\}$$

Step 1: Eliminate Implications

Given:

$$\forall X \left\{ p(X) \rightarrow \left[\forall Y \left[p(Y) \rightarrow p(f(X,Y)) \right] \wedge \neg \left[\forall Y (\neg q(X,Y) \wedge p(Y)) \right] \right] \right\}$$

Eliminate implications: replace $A \rightarrow B$ with $\neg A \vee B$

Obtain:

$$\forall X \left\{ \neg p(X) \vee \left[\forall Y \left[\neg p(Y) \vee p(f(X,Y)) \right] \wedge \neg \left[\forall Y (\neg q(X,Y) \wedge p(Y)) \right] \right] \right\}$$

Step 2: Move Negation Inwards

Given:

$$\forall X\left\{\neg p(X) \vee \left[\forall Y\left[\neg p(Y) \vee p(f(X,Y))\right] \wedge \neg\left[\forall Y(\neg q(X,Y) \wedge p(Y))\right]\right]\right\}$$

Move negation inwards and simplify double negations

$$\begin{array}{rcl}
\neg(\neg A) & \text{becomes} & A \\
\neg(A \wedge B) & \text{becomes} & (\neg A \vee \neg B) \\
\neg(A \vee B) & \text{becomes} & (\neg A \wedge \neg B) \\
\neg\forall X(A) & \text{becomes} & \exists X(\neg A) \\
\neg\exists X(A) & \text{becomes} & \forall X(\neg A)
\end{array}$$

Obtain:

$$\forall X\left\{\neg p(X) \vee \left[\forall Y\left[\neg p(Y) \vee p(f(X,Y))\right] \wedge \left[\exists Y(q(X,Y) \vee \neg p(Y))\right]\right]\right\}$$

Step 3: Standardize Variables

Given:

$$\forall X \left\{ \neg p(X) \vee \left[\forall Y \left[\neg p(Y) \vee p(f(X,Y)) \right] \wedge \left[\exists Y (q(X,Y) \vee \neg p(Y)) \right] \right] \right\}$$

Rename variables so that each quantified variable is unique

Obtain:

$$\forall X \left\{ \neg p(X) \vee \left[\forall Y \left[\neg p(Y) \vee p(f(X,Y)) \right] \wedge \left[\exists Z (q(X,Z) \vee \neg p(Z)) \right] \right] \right\}$$

Step 4: Skolemize

Given:

$$\forall X \left\{ \neg p(X) \vee \left[\forall Y \left[\neg p(Y) \vee p(f(X,Y)) \right] \wedge \left[\exists Z (q(X,Z) \vee \neg p(Z)) \right] \right] \right\}$$

"Skolemize": remove all existential quantifiers $\exists$ by introducing **new function symbols** in place of the formerly-quantified variables

Obtain:

$$\forall X \left\{ \neg p(X) \vee \left[\forall Y \left[\neg p(Y) \vee p(f(X,Y)) \right] \wedge \left[q(X,g(X)) \vee \neg p(g(X)) \right] \right] \right\}$$

Skolemization Explained

Consider an example:

$$\exists Y(\mathit{elephant}(Y) \wedge \mathit{friendly}(Y))$$

- This states that there is some individual (a binding for Y) that is both an elephant and friendly
- To remove the existential quantifier, we **invent** a name for this entity, let's say a. This is a **new** constant symbol, not equal to any previous constant symbols. We get a logically equivalent statement:

$$\mathit{elephant}(a) \wedge \mathit{friendly}(a)$$

- This is saying exactly the same thing, since we know nothing about this new constant apart from the fact that it exists and has to be bound to some individual

Skolemization Explained

- It is essential that the introduced symbol a is **new**. Otherwise, we might know something about a from the KB
- If the KB had something to say about the constant a, we would be asserting more than the existential did about that individual(s) hiding under the name Y
- In the original quantified formula, we know nothing about the variable Y except what was being asserted by the formula itself

Skolemization Explained

A less trivial example:

$$\forall X \exists Y (loves(X, Y))$$

- This states that *for every* X there is some Y that loves X — could be a different Y for each X
- Replacing the existential by a new constant won't work

  $$\forall X (loves(X, a))$$

  because now there is **one particular individual** a loved by every X
- To properly convert existential quantifiers which are inside the scope of universal quantifiers, we must use **functions** and not just dumb constants

Skolemization Explained

- We must use a function which takes as an argument (i.e., depends on) every universally quantified variable **that scopes the existential**
- In our example, Y is inside the scope of $\forall X$ ("X scopes Y"), so we must replace the existential Y by a function of X:

  $$\forall X(\mathit{loves}(X, g(X))),$$

  where g is a **new** function symbol.
- Now, the formula asserts that for every X there is some individual (as given by $g(X)$) that X loves. Since g is a new symbol, it could be interpreted arbitrarily, so $g(X)$ could be different for each binding of X.

Skolemization: Some More Examples

$$\forall X \forall Y \forall Z \exists W.\ r(X, Y, Z, W)$$

becomes

$$\forall X \forall Y \forall Z.\ r(X, Y, Z, h_1(X, Y, Z))$$

$$\forall X \forall Y \exists W.\ r(X, Y, g(W))$$

becomes

$$\forall X \forall Y.\ r(X, Y, g(h_2(X, Y)))$$

$$\forall X \forall Y \exists W \forall Z.\ r(X, Y, W) \wedge q(Z, W)$$

becomes

$$\forall X \forall Y \forall Z.\ r(X, Y, h_3(X, Y)) \wedge q(Z, h_3(X, Y))$$

Step 5: Convert to Prenex Normal Form

Given:

$$\forall X \left\{ \neg p(X) \vee \left[\forall Y \left[\neg p(Y) \vee p(f(X, Y)) \right] \wedge \left[q(X, g(X)) \vee \neg p(g(X)) \right] \right] \right\}$$

Bring all quantifiers to the outside (front). At this point, we only have universals left, and each quantifies a differently-named variable

Obtain:

$$\forall X \forall Y \left\{ \neg p(X) \vee \left[\left[\neg p(Y) \vee p(f(X, Y)) \right] \wedge \left[q(X, g(X)) \vee \neg p(g(X)) \right] \right] \right\}$$

Step 6: Conjunctions Over Disjunctions

Given:

$$\forall X \forall Y \left\{ \neg p(X) \vee \left[\left[\neg p(Y) \vee p(f(X, Y)) \right] \wedge \left[q(X, g(X)) \vee \neg p(g(X)) \right] \right] \right\}$$

Apply the distributive law: $A \vee (B \wedge C)$ becomes $(A \vee B) \wedge (A \vee C)$

Obtain:

$$\forall X \forall Y \Big\{ \left[\neg p(X) \vee (\neg p(Y) \vee p(f(X, Y))) \right] \\ \wedge \left[\neg p(X) \vee (q(X, g(X)) \vee \neg p(g(X))) \right] \Big\}$$

Step 7: Flatten

Given:

$$\forall X \forall Y \Big\{ \big[\neg p(X) \vee (\neg p(Y) \vee p(f(X, Y)))\big] \wedge \big[\neg p(X) \vee (q(X, g(X)) \vee \neg p(g(X)))\big] \Big\}$$

Flatten nested conjunctions and disjunctions: $(A \vee (B \vee C))$ becomes $(A \vee B \vee C)$

Obtain:

$$\forall X \forall Y \Big\{ \big[\neg p(X) \vee \neg p(Y) \vee p(f(X, Y))\big] \wedge \big[\neg p(X) \vee q(X, g(X)) \vee \neg p(g(X))\big] \Big\}$$

Step 8: Convert to Clauses

Given:

$$\forall X \forall Y \left\{ \left[\neg p(X) \vee \neg p(Y) \vee p(f(X,Y)) \right] \right.$$
$$\left. \wedge \left[\neg p(X) \vee q(X, g(X)) \vee \neg p(g(X)) \right] \right\}$$

Remove quantifiers and break apart conjunctions (this is purely notational, we are not changing the formula any more. The removed symbols become implicit)

Obtain:

$$\neg p(X) \vee \neg p(Y) \vee p(f(X,Y))$$
$$\neg p(X) \vee q(X, g(X)) \vee \neg p(g(X))$$

We are now in clausal form.

End of Lecture

$$(\neg p(X), \neg p(Y), p(f(X, Y)))$$
$$(\neg p(X), q(X, g(X)), \neg p(g(X)))$$

- Observe: we now have variables in the clauses!
- Next lecture: how resolution handles this
- (Unification)