L21: First-order Logic Proof

AIMA4e: Chapter 7.5, 9.1, 9.2, 9.5

What you should know after this lecture

- First-order resolution theorem proving

Syntactic proof

Recall, a proof procedure takes two sentences, α and β, and tells you whether it can prove β from α:

$$\alpha \vdash \beta$$

Proof procedure is

- sound iff for all α, β, if $\alpha \vdash \beta$ then $\alpha \models \beta$
- complete iff for all α, β, if $\alpha \models \beta$ then $\alpha \vdash \beta$

We have looked at proof procedures that operate via enumerating models. But that is incomplete and/or inefficient in many cases. So, we will look at purely syntactic proof, that operates entirely on logical sentences.

One proof strategy: resolution refutation

To prove $\alpha \models \beta$:

- Write α as one or more premises
- Inference rules tell you what you can add to your proof given what you already have. Logic is monotonic.
- When the rules have allowed you to write down β, then you're done.

Proof by refutation:

- To prove $\alpha \models \beta$
- Instead show that $\alpha \wedge \neg\beta \models$ **False**

Inference rules:

- Lots of interesting proof systems (sets of inference rules)
- We would like one that is sound and complete: $(\alpha \vdash \beta) \equiv (\alpha \models \beta)$
- Refutation using the resolution inference rule is sound and complete!!

Propositional resolution: reminder

General inference rule form: If you have α and β written down in your proof, you can now write γ.

$$\frac{\alpha \quad \beta}{\gamma}$$

Modus Ponens:

$$\frac{P \Rightarrow Q \quad P}{Q}$$

Propositional Resolution:

$$\frac{(P \lor Q_1 \lor \ldots, \lor Q_n) \quad (\neg P \lor R_1 \lor \ldots \lor R_m)}{(Q_1 \lor \ldots \lor Q_n \lor R_1 \lor \ldots \lor R_m)}$$

Clausal form

Resolution requires sentences in first-order clausal form.

1. Rename variables so that they are all distinct.
2. Convert implications into disjunctions.
3. Push negations all the way in, using FO DeMorgan:
   $\neg\exists x.\alpha \equiv \forall x.\neg\alpha$ and $\neg\forall x.\alpha \equiv \exists x.\neg\alpha$
4. Move all quantifiers to the front, maintaining their order.
5. Replace every existentially quantified variable with a Skolem function of any universally quantified variables that come before it.
6. Drop the universal quantifiers.
7. Convert to CNF.

Clausal form practice

Every dog has its day.

$$\forall x.Dog(x) \Rightarrow \exists y.Day(y) \wedge Has(x, y)$$
$$\forall x.\neg Dog(x) \vee \exists y.Day(y) \wedge Has(x, y)$$
$$\forall x.\exists y.\neg Dog(x) \vee (Day(y) \wedge Has(x, y))$$
$$\forall x.\neg Dog(x) \vee (Day(f_1(x)) \wedge Has(x, f_1(x)))$$
$$\neg Dog(x) \vee (Day(f_1(x)) \wedge Has(x, f_1(x)))$$
$$(\neg Dog(x) \vee Day(f_1(x))) \wedge (\neg Dog(x) \vee Has(x, f_1(x)))$$

There is at least one dog!

$\exists x.Dog(x)$

$Dog(f_2)$

There are no days.

$\neg \exists x.Day(x)$

$\forall x.\neg Day(x)$

$\neg Day(x)$

Unification: matching literals

Returns substitution: $\{v_1/t_1, \ldots, v_k/t_k\}$; variables v_i terms t_i. The most general substitution that makes α and β equal.

UNIFY(α, β, θ)

if $\theta =$'fail' **return** 'fail'
if $\alpha = \beta$ **return** θ
if IS-VAR(α) **return** UNIFY-VAR(α, β, θ)
if IS-VAR(β) **return** UNIFY-VAR(β, α, θ)
if STRUCT(α) and STRUCT(β):
  return UNIFY$(\alpha[1:], \beta[1:], \text{UNIFY}(\alpha[0], \beta[0], \theta))$
else return 'fail'

UNIFY-VAR(α, β, θ)

if $\{\alpha/\gamma\} \in \theta$ **return** UNIFY(γ, β, θ)
if $\{\beta/\gamma\} \in \theta$ **return** UNIFY(γ, α, θ)
if OCCURS(α, β) **return** 'fail'
else return $\theta \cup \{\alpha/\beta\}$

Unification examples

α	β	θ
$A(B, C)$	$A(x, y)$	$\{x/B, y/C\}$
$A(x, f(D, x))$	$A(E, f(D, y))$	$\{x/E, y/E\}$
$A(x, y)$	$A(f(C, y), z)$	$\{x/f(C, y), y/z\}$
$P(A, x, f(g(y)))$	$P(y, f(z), f(z)),$	$\{y/A, x/f(z), z/g(y)\}$
$P(x, g(f(A)), f(x))$	$P(f(y), z, y)$	fail
$P(x, f(y))$	$P(z, g(w))$	fail
$P(x)$	$Q(x)$	fail

Resolution!

$$\frac{(l_1 \vee \ldots \vee l_n) \quad (m_1 \vee \ldots \vee m_k)}{\textsc{subst}(\theta, l_2 \vee \ldots \vee l_n \vee m_2 \vee \ldots \vee m_k)}$$

where $\textsc{unify}(l_1, \neg m_1) = \theta$.

Plus one more trick called factoring: basically, internal unification.

Theorem: Resolution plus factoring is refutation complete.

If you have equality, you need one more trick: paramodulation.

Dog days

Do these two sentences

$$\forall x. Dog(x) \Rightarrow \exists y. Day(y) \land Has(x, y)$$
$$\exists x. Dog(x)$$

entail

$$\exists x. Day(x)$$

Prove it!

Write down α and $\neg\beta$ in clausal form. Try to prove **False**.

1. $\neg Dog(x) \vee Day(f_1(x))$
2. $\neg Dog(x) \vee Has(x, f_1(x))$
3. $Dog(f_2)$
4. $\neg Day(x)$
5. $Day(f_1(f_2))$ — $1, 3$ $\{x/f_2\}$
6. **False** — $4, 5$ $\{x/f_1(f_2)\}$

So, yes, if there's a dog, there's a day!

Horn clauses

A Horn clause is a clause (disjunction of literals) with exactly one positive literal. Looks like

$$\alpha \wedge \beta \wedge \gamma \Rightarrow \delta$$

Datalog: Horn clauses with no function symbols. More efficient inference. Decidable.

Prolog: Horn clauses. Depth-first backward chaining. Basis of logic programming which then adds extra tricks for handling negation, equality, and even side-effects.

Completeness and decidability

Goedel's Completeness Theorem: There exists a complete proof system for FOL.

Robinson's Completeness Theorem: Resolution is a refutation complete proof system for FOL.

FOL is semi-decidable: if $\alpha \models \beta$ then eventually resolution refutation will find a contradiction. But if not, it might run forever!

Goedel's First Incompleteness Theorem: There is no consistent, complete proof system for FOL with arithmetic ($+$ and $\times$).

Arithmetic allows you to construct code-names for sentences within the logic, so that $P =$ "P is not provable". Then

- If P is true: P is not provable (incomplete)
- If P is false: P is provable (inconsistent)