Review: ingredients of a logic

Syntax: defines a set of valid formulas (Formulas)
Example: Rain ∧ Wet

Semantics: for each formula, specify a set of models (assignments/configurations of the world)
Example: from Rain ∧ Wet, derive Rain

Inference rules: given f, what new formulas g can be added that are guaranteed to follow \( \frac{f}{g} \)?
Example: from Rain ∧ Wet, derive Rain

- Logic provides a formal language to talk about the world.
- The valid sentences in the language are the logical formulas, which live in syntax-land.
- In semantics-land, a model represents a possible configuration of the world. An interpretation function connects syntax and semantics. Specifically, it defines, for each formula f, a set of models \( M(f) \).

Review: schema

Syntax

- formula
- Inference rules

Semantics

- models

Review: inference task

Input:
Knowledge base KB (e.g., \{Rain, Rain → Wet\})
Query formula f (e.g., Wet)

Output:
Whether KB entails f (KB \( \models f \)) (e.g., yes)
\( (KB \models f \iff M(f) \supseteq M(KB)) \)

- A knowledge base is a set of formulas we know to be true. Semantically the KB represents the conjunction of the formulas.
- The central goal of logic is inference: to figure out whether a query formula is entailed by, contradictory with, or contingent on the KB (these are semantic notions defined by the interpretation function).
Review: inference algorithm

Inference algorithm:
\[ \text{KB} \xrightarrow{\text{repeatedly apply inference rules}} f \]

Definition: modus ponens inference rule
\[ p_1, \ldots, p_k, (p_1 \land \cdots \land p_k) \rightarrow q \]
\[ q \]

Desiderata: soundness and completeness

Review: formulas

Propositional logic: any legal combination of symbols

\[(\text{Rain} \land \text{Snow}) \rightarrow (\text{Traffic} \lor \text{Peaceful}) \land \text{Wet}\]

Propositional logic with only Horn clauses: restricted

\[(\text{Rain} \land \text{Snow}) \rightarrow \text{Traffic}\]

Approaches

<table>
<thead>
<tr>
<th>Formulas allowed</th>
<th>Inference rule</th>
<th>Complete?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional logic (only Horn clauses)</td>
<td>modus ponens</td>
<td>yes</td>
</tr>
<tr>
<td>Propositional logic</td>
<td>modus ponens</td>
<td>no</td>
</tr>
<tr>
<td>Propositional logic</td>
<td>resolution</td>
<td>yes</td>
</tr>
</tbody>
</table>

- The unique thing about having a logical language is that we can also perform inference directly on syntax by applying inference rules, rather than always appealing to semantics (and performing model checking there).
- We would like the inference algorithm to be both sound (not derive any false formulas) and complete (derive all true formulas). Soundness is easy to check, completeness is harder.

- Whether a set of inference rules is complete depends on what the formulas are. Last time, we looked at two logical languages: propositional logic and propositional logic restricted to Horn clauses (essentially formulas that look like \( p_1 \land \cdots \land p_k \rightarrow q \)), which intuitively can only derive positive information.

- We saw that if our logical language was restricted to Horn clauses, then modus ponens alone was sufficient for completeness. For general propositional logic, modus ponens is insufficient.
- In this lecture, we'll see that a more powerful inference rule, resolution, is complete for all of propositional logic.
Roadmap
Resolution in propositional logic
First-order logic

- Modus ponens can only deal with Horn clauses, so let’s see why Horn clauses are limiting. We can equivalently write implication using negation and disjunction. Then it’s clear that Horn clauses are just disjunctions of literals where there is at most one positive literal and zero or more negative literals. The negative literals correspond to the propositional symbols on the left side of the implication, and the positive literal corresponds to the propositional symbol on the right side of the implication.
- If we rewrite modus ponens, we can see a “canceling out” intuition emerging. To make the intuition a bit more explicit, remember that, to respect soundness, we require \( \{ A, \neg A \lor C \} \models C \); this is equivalent to: if \( A \land (\neg A \lor C) \) is true, then \( C \) is also true. This is clearly the case.
- But modus ponens cannot operate on general clauses.

- Let’s try to generalize modus ponens by allowing it to work on general clauses. This generalized inference rule is called resolution, which was invented in 1965 by John Alan Robinson.
- The idea behind resolution is that it takes two general clauses, where one of them has some propositional symbol \( p \) and the other clause has its negation \( \neg p \), and simply takes the disjunction of the two clauses with \( p \) and \( \neg p \) removed. Here, \( f_1, \ldots, f_n, g_1, \ldots, g_m \) are arbitrary literals.

Horn clauses and disjunction

Written with implication Written with disjunction
\[ A \rightarrow C \] \[ \neg A \lor C \]
\[ A \land B \rightarrow C \] \[ \neg A \lor \neg B \lor C \]

- Literal: either \( p \) or \( \neg p \), where \( p \) is a propositional symbol
- Clause: disjunction of literals
- Horn clauses: at most one positive literal

Modus ponens (rewritten):
\[ \frac{A \quad \neg A \lor C}{C} \]

- Intuition: cancel out \( A \) and \( \neg A \)

Resolution [Robinson, 1965]

General clauses have any number of literals:
\[ \neg A \lor B \lor \neg C \lor D \lor \neg E \lor F \]

Example: resolution inference rule
\[
\begin{align*}
\text{Rain} \lor \text{Snow}, & \quad \neg \text{Snow} \lor \text{Traffic} \\
& \quad \text{Rain} \lor \text{Traffic}
\end{align*}
\]

Definition: resolution inference rule
\[
\frac{f_1 \lor \cdots \lor f_n \lor p, \quad \neg p \lor g_1 \lor \cdots \lor g_m}{f_1 \lor \cdots \lor f_n \lor g_1 \lor \cdots \lor g_m}
\]

Soundness of resolution

\[
\begin{align*}
\text{Resolution rule} & \\
\text{Rain} \lor \text{Snow}, & \quad \neg \text{Snow} \lor \text{Traffic} \\
\text{Rain} & \lor \text{Traffic}
\end{align*}
\]

\[
\begin{align*}
\mathcal{M}(\text{Rain} \lor \text{Snow}) \cap \mathcal{M}(\neg \text{Snow} \lor \text{Traffic}) & \subseteq \mathcal{M}(\text{Rain} \lor \text{Traffic}) \\
\end{align*}
\]

\[
\begin{align*}
\text{Rain} & \quad \text{Traffic} \\
\text{Snow} & \quad 0 \quad 1 \\
0 & \quad 1 \quad 1.0 \quad 0 \quad 0.0 \\
1 & \quad 1.4 \quad 0.4 \quad 0.1 \quad 1.1
\end{align*}
\]

Sound!
Why is resolution logically sound? We can verify the soundness of resolution by checking its semantic interpretation. Indeed, the intersection of the models of \( f \) and \( g \) is a subset of models of \( f \lor g \).

Conversion to CNF: general

Conversion rules:

- Eliminate \( \leftrightarrow \): \( f \leftrightarrow g \)
  \( (f \rightarrow g) \land (g \rightarrow f) \)
- Eliminate \( \rightarrow \): \( f \rightarrow g \)
  \( \lnot f \lor g \)
- ... \( \land \lnot g \)
- Eliminate double negation: \( \lnot \lnot f \)
  \( f \)
- Distribute \( \lor \) over \( \land \): \( f \lor (g \land h) \)
  \( (f \lor g) \land (f \lor h) \)

But so far, we've only considered clauses, which are disjunctions of literals. Surely this can't be all of propositional logic... But it turns out it actually is in the following sense.

A conjunction of clauses is called a CNF formula, and every formula in propositional logic can be converted into an equivalent CNF formula. Given a CNF formula, we can toss each of its clauses into the knowledge base.

But why can every formula be put in CNF?

The answer is by construction. There is a six-step procedure that takes any propositional formula and turns it into CNF. Here is an example of how it works (only four of the six steps apply here).

Conjunctive normal form

So far: resolution only works on clauses...but that's enough!

**Definition**: conjunctive normal form (CNF)

A CNF formula is a conjunction of clauses.

Example: \((A \lor B \lor \lnot C) \land (\lnot B \lor D)\)

Equivalent: knowledge base where each formula is a clause

**Proposition**: conversion to CNF

Every formula \( f \) in propositional logic can be converted into an equivalent CNF formula \( f' \):

\[ M(f) = M(f') \]

Conversion to CNF: example

Initial formula:

\((\text{Summer} \rightarrow \text{Snow}) \rightarrow \text{Bizzare}\)

Remove implication \( \rightarrow \):

\(\lnot (\lnot \text{Summer} \lor \text{Snow}) \lor \text{Bizzare}\)

Push negation \( \lnot \) inwards (de Morgan):

\((\lnot \lnot \text{Summer} \land \lnot \text{Snow}) \lor \text{Bizzare}\)

Remove double negation:

\((\text{Summer} \land \lnot \text{Snow}) \lor \text{Bizzare}\)

Distribute \( \lor \) over \( \land \):

\((\text{Summer} \lor \text{Bizzare}) \land (\lnot \text{Snow} \lor \text{Bizzare})\)

Conversion to CNF: general

Conversion rules:

- Eliminate \( \leftrightarrow \): \( f \leftrightarrow g \)
  \( (f \rightarrow g) \land (g \rightarrow f) \)
- Eliminate \( \rightarrow \): \( f \rightarrow g \)
  \( \lnot f \lor g \)
- ... \( \land \lnot g \)
- Eliminate double negation: \( \lnot \lnot f \)
  \( f \)
- Distribute \( \lor \) over \( \land \): \( f \lor (g \land h) \)
  \( (f \lor g) \land (f \lor h) \)
Here are the general rules that convert any formula to CNF. First, we try to reduce everything to negation, conjunction, and disjunction.

Next, we try to push negation inwards so that they sit on the propositional symbols (forming literals). Note that when negation gets pushed inside, it flips conjunction to disjunction, and vice-versa.

Finally, we distribute so that the conjunctions are on the outside, and the disjunctions are on the inside.

Note that each of these operations preserves the semantics of the logical form (remember there are many formula that map to the same set of models). This is in contrast with most inference rules, where the conclusion is more general than the conjunction of the premises.

Also, when we apply a CNF rewrite rule, we replace the old formula with the new one, so there is no blow-up in the number of formulas. This is in contrast to applying general inference rules. An analogy: conversion to CNF does simplification in the context of full inference, just like AC-3 does simplification in the context of backtracking search.

Resolution algorithm
Recall: entailment and contradiction $\iff$ satisfiability
$\text{KB} \models f$ $\iff$ $\text{KB} \cup \{\neg f\}$ is unsatisfiable

$\text{KB} \models \neg f$ $\iff$ $\text{KB} \cup \{f\}$ is unsatisfiable

Algorithm: resolution-based inference
- Convert all formulas into CNF.
- Repeatedly apply resolution rule.
- Return unsatisfiable iff derive false.

Resolution: example
Knowledge base (is it satisfiable?):
$\text{KB} = \{A \to (B \lor C), A, \neg B, \neg C\}$

Convert to CNF:
$\text{KB} = \{\neg A \lor B \lor C, A, \neg B, \neg C\}$

Repeatedly apply resolution rule:

Unsatisfiable!

Time complexity
Definition: modus ponens inference rule
$p_1, \ldots, p_k, (p_1 \land \cdots \land p_k) \to q$

- Each rule application adds clause with one propositional symbol
  $\Rightarrow$ linear time

Definition: resolution inference rule
$f_1 \lor \cdots \lor f_m \lor p, \neg p \lor q_1 \lor \cdots \lor q_n$

- Each rule application adds clause with many propositional symbols
  $\Rightarrow$ exponential time
• There we have it — a sound and complete inference procedure for all of propositional logic (although we didn’t prove completeness). But what do we have to pay computationally for this increase?
• If we only have to apply modus ponens, each propositional symbol can only get added once, so with the appropriate algorithm (forward chaining), we can apply all necessary modus ponens rules in linear time.
• But with resolution, we can end up adding clauses with many propositional symbols, and possibly any subset of them! Therefore, this can take exponential time.

Limitations of propositional logic

Alice and Bob both know arithmetic.
AliceKnowsArithmetic ∧ BobKnowsArithmetic

All students know arithmetic.
AliceIsStudent → AliceKnowsArithmetic
BobIsStudent → BobKnowsArithmetic

Every even integer greater than 2 is the sum of two primes.

To summarize, we can either content ourselves with the limited expressivity of Horn clauses and obtain an efficient inference procedure (via modus ponens).
• If we wanted the expressivity of full propositional logic, then we need to use resolution and thus pay more.

• If the goal of logic is to be able to express facts in the world in a compact way, let us ask ourselves if propositional logic is enough.
• Some facts can be expressed in propositional logic, but it is very clunky, having to instantiate many different formulas. Others simply can’t be expressed at all, because we would need to use an infinite number of formulas.

Summary

Horn clauses any clauses
modus ponens resolution
linear time exponential time
less expressive more expressive

Roadmap

Resolution in propositional logic
First-order logic
Limitations of propositional logic

All students know arithmetic.

AliceIsStudent → AliceKnowsArithmetic
BobIsStudent → BobKnowsArithmetic
...

Propositional logic is very clunky. What’s missing?

- **Objects and relations**: propositions (e.g., AliceKnowsArithmetic) have more internal structure (alice, Knows, arithmetic)
- **Quantifiers and variables**: *all* is a quantifier which applies to each person, don’t want to enumerate them all...

What’s missing? The key conceptual observation is that the world is not just a bunch of atomic facts, but that each fact is actually made out of objects and relations between those objects.

Once facts are decomposed in this way, we can use quantifiers and variables to implicitly define a huge (and possibly infinite) number of facts with one compact formula. Again, where logic excels is the ability to represent complex things via simple means.

First-order logic

- **Syntax**: formulas, variables, quantifiers
- **Semantics**: models
- **Inference rules**: rules to operate on the syntax

We will now introduce **first-order logic**, which will address the representational limitations of propositional logic.

Remember to define a logic, we need to talk about its syntax, its semantics (interpretation function), and finally inference rules that we can use to operate on the syntax.

First-order logic: examples

Alice and Bob both know arithmetic.

Knows(alice, arithmetic) ∧ Knows(bob, arithmetic)

All students know arithmetic.

∀x Student(x) → Knows(x, arithmetic)
Syntax of first-order logic

Terms (refer to objects):
- Constant symbol (e.g., arithmetic)
- Variable (e.g., \(x\))
- Function of terms (e.g., \(\text{Sum}(3, x)\))

Formulas (refer to truth values):
- Atomic formulas (atoms): predicate applied to terms (e.g., \(\text{Knows}(x, \text{arithmetic})\))
- Connectives applied to formulas (e.g., \(\text{Student}(x) \to \text{Knows}(x, \text{arithmetic})\))
- Quantifiers applied to formulas (e.g., \(\forall x \text{Student}(x) \to \text{Knows}(x, \text{arithmetic})\))

Quantifiers

Universal quantification (\(\forall\)):

Think conjunction: \(\forall x P(x)\) is like \(P(A) \land P(B) \land \cdots\)

Existential quantification (\(\exists\)):

Think disjunction: \(\exists x P(x)\) is like \(P(A) \lor P(B) \lor \cdots\)

Some properties:
- \(\neg \forall x P(x)\) equivalent to \(\exists x \neg P(x)\)
- \(\forall x \exists y \text{Knows}(x, y)\) different from \(\exists y \forall x \text{Knows}(x, y)\)

Natural language quantifiers

Universal quantification (\(\forall\)):

*Every student knows arithmetic.*

\(\forall x \text{Student}(x) \to \text{Knows}(x, \text{arithmetic})\)

Existential quantification (\(\exists\)):

*Some student knows arithmetic.*

\(\exists x \text{Student}(x) \land \text{Knows}(x, \text{arithmetic})\)

\[
\text{Note the different connectives!}
\]
Some examples of first-order logic

**There is some course that every student has taken.**

\[ \exists y \text{Course}(y) \land \left[ \forall x \text{Student}(x) \rightarrow \text{Takes}(x, y) \right] \]

**Every even integer greater than 2 is the sum of two primes.**

\[ \forall x \text{EvenInt}(x) \land \text{Greater}(x, 2) \rightarrow \exists y \exists z \text{Sum}(x, y, z) \land \text{Prime}(y) \land \text{Prime}(z) \]

**If a student takes a course and the course covers a concept, then the student knows that concept.**

\[ \forall x \forall y \forall z \left( \text{Student}(x) \land \text{Takes}(x, y) \land \text{Course}(y) \land \text{Covers}(y, z) \right) \rightarrow \text{Knows}(x, z) \]

---

**First-order logic**

**Syntax**

- formula
- models

**Semantics**

- Inference rules

---

**Models in first-order logic**

Recall a model represents a possible situation in the world.

**Propositional logic:** Model \( w \) maps propositional symbols to truth values.

\[ w = \{ \text{AliceKnowsArithmetic} : 1, \text{BobKnowsArithmetic} : 0 \} \]

**First-order logic:** ?

---

- Let’s do some more examples of converting natural language to first-order logic. Remember the connectives associated with existential and universal quantification!
- Note that some English words such as a can trigger both universal or existential quantification, depending on context. In *A student took CS221*, we have existential quantification, but in *if a student takes CS221, ...*, we have universal quantification.
- Formal logic clears up the ambiguities associated with natural language.

- So far, we’ve only presented the syntax of first-order logic, although we’ve actually given quite a bit of intuition about what the formulas mean. After all, it’s hard to talk about the syntax without at least a hint of semantics for motivation.
- Now let’s talk about the formal semantics of first-order logic.

- Recall that a model in propositional logic was just an assignment of truth values to propositional symbols.
- A natural candidate for a model in first-order logic would then be an assignment of truth values to grounded atomic formula (those formulas whose terms are constants as opposed to variables). This is almost right, but doesn’t talk about the relationship between constant symbols.
Graph representation of a model

If only have unary and binary relations, a model $w$ can be represented as a directed graph:

- Nodes are objects, labeled with constant symbols
- Directed edges are binary relations, labeled with predicate symbols; unary relations are additional node labels

Models in first-order logic

**Definition: model in first-order logic**

A model $w$ in first-order logic maps:

- constant symbols to objects
  \[ w(\text{alice}) = o_1, w(\text{bob}) = o_2, w(\text{arithmetic}) = o_3 \]
- predicate symbols to tuples of objects
  \[ w(\text{Knows}) = \{(o_1, o_3), (o_2, o_3), \ldots \} \]

A restriction on models

*John and Bob are students.*

$\text{Student(john)} \land \text{Student(bob)}$

- **Unique names assumption**: Each object has at most one constant symbol. This rules out $w_2$.
- **Domain closure**: Each object has at least one constant symbol. This rules out $w_3$.

Point:

constant symbol \hspace{1cm} object
Propositionalization

If one-to-one mapping between constant symbols and objects (unique names and domain closure), first-order logic is syntactic sugar for propositional logic:

Knowledge base in first-order logic

\[
\text{Student}(\text{alice}) \land \text{Student}(\text{bob}) \\
\forall x \text{ Student}(x) \rightarrow \text{Person}(x) \\
\exists x \text{ Student}(x) \land \text{Creative}(x)
\]

Knowledge base in propositional logic

\[
\text{Student}(\text{alice}) \land \text{Student}(\text{bob}) \\
\text{Student}(\text{alice}) \rightarrow \text{Person}(\text{alice}) \land \text{Student}(\text{bob}) \rightarrow \text{Person}(\text{bob}) \\
\text{Student}(\text{alice}) \land \text{Creative}(\text{alice}) \lor \text{Student}(\text{bob}) \land \text{Creative}(\text{bob})
\]

Point: use any inference algorithm for propositional logic!

First-order logic

Syntax

Semantics

Inference rules

Definite clauses

\[
\forall x \forall y \forall z (\text{Takes}(x, y) \land \text{Covers}(y, z)) \rightarrow \text{Knows}(x, z)
\]

Note: if propositionalize, get one formula for each value to \((x, y, z)\), e.g., (alice, cs221, mdp)

Definition: definite clause (first-order logic)

A definite clause has the following form:

\[
\forall x_1 \cdots \forall x_n (a_1 \land \cdots \land a_k) \rightarrow b
\]

for variables \(x_1, \ldots, x_n\) and atomic formulas \(a_1, \ldots, a_k, b\) (which contain those variables).
Modus ponens (first attempt)

Definition: modus ponens (first-order logic)

\[ a_1, \ldots, a_k \forall x_1 \cdots \forall x_n (a_1 \land \cdots \land a_k) \rightarrow b \]

Setup:
Given \( P(\text{alice}) \) and \( \forall x \ P(x) \rightarrow Q(x) \).

Problem:
Can’t infer \( Q(\text{alice}) \) because \( P(x) \) and \( P(\text{alice}) \) don’t match!

Solution: substitution and unification

Substitution
Subst\([^x/\text{alice}], P(x) \]= \( P(\text{alice}) \)

Subst\([^x/\text{alice}, y/z], P(x) \land K(x, y) \]= \( P(\text{alice}) \land K(\text{alice}, z) \)

Definition: Substitution
A substitution \( \theta \) is a mapping from variables to terms.
Subst\([^\theta, f] \) returns the result of performing substitution \( \theta \) on \( f \).

Unification

Unify\([\text{Knows}(\text{alice}, \text{arithmetic}), \text{Knows}(x, \text{arithmetic})]\)= \{x/\text{alice}\}

Unify\([\text{Knows}(\text{alice}, y), \text{Knows}(x, z)\]= \{x/\text{alice}, y/z\}

Unify\([\text{Knows}(\text{alice}, y), \text{Knows}(\text{bob}, z)\]= \text{fail}

Unify\([\text{Knows}(\text{alice}, y), \text{Knows}(x, P(x))\]= \{x/\text{alice}, y/F(\text{alice})\}

Definition: Unification
Unification takes two formulas \( f \) and \( g \) and returns a substitution \( \theta \) which is the most general unifier:
Unify\([f, g] = \theta \) such that Subst\([^\theta, f] = \text{Subst}[\theta, g] \)
or "fail" if no such \( \theta \) exists.
Modus ponens

**Definition: modus ponens (first-order logic)**

\[ a_1', \ldots, a_k' \quad \forall x_1 \cdots \forall x_n (a_1 \land \cdots \land a_k) \rightarrow b \]

Get most general unifier \( \theta \) on premises:

- \( \theta = \text{Unify} [a_1' \land \cdots \land a_k', a_1 \land \cdots \land a_k] \)

Apply \( \theta \) to conclusion:

- \( \text{Subst} [\theta, b] = b' \)

---

**Modus ponens example**

**Example: modus ponens in first-order logic**

**Premises:**

- Takes(alice, cs221)
- Covers(cs221, mdp)
- \( \forall x \forall y \forall z \text{Takes}(x, y) \land \text{Covers}(y, z) \rightarrow \text{Knows}(x, z) \)

**Conclusion:**

- \( \theta = \{x/\text{alice}, y/cs221, z/\text{mdp}\} \)
- Derive Knows(alice, mdp)

---

**Complexity**

\( \forall x \forall y \forall z P(x, y, z) \)

- Each application of Modus ponens produces an atomic formula.

- If no function symbols, number of atomic formulas is at most

  \( (\text{num-constant-symbols})^{(\text{maximum-predicate-arity})} \)

- If there are function symbols (e.g., \( F \)), then infinite...

  \( Q(a) \quad Q(F(a)) \quad Q(F(F(a))) \quad Q(F(F(F(a)))) \quad \cdots \)

---

- Having defined substitution and unification, we are in position to finally define the modus ponens rule for first-order logic. Instead of performing an exact match, we instead perform a unification, which generates a substitution \( \theta \). Using \( \theta \), we can generate the conclusion \( b' \) on the fly.

- Note the significance here: the rule \( a_1' \land \cdots \land a_k' \rightarrow b \) can be used in a myriad ways, but Unify identifies the appropriate substitution, so that it can be applied to the conclusion.

- Here’s a simple example of modus ponens in action. We bind \( x, y, z \) to appropriate objects (constant symbols), which is used to generate the conclusion Knows(alice, mdp).

- In propositional logic, modus ponens was considered efficient, since in the worst case, we generate each propositional symbol.

- In first-order logic, though, we typically have many more atomic formulas in place of propositional symbols, which leads to a potentially exponentially number of atomic formulas, or worse, with function symbols, there might be an infinite set of atomic formulas.
Complexity

**Theorem: completeness**

Modus ponens is complete for first-order logic with only Horn clauses.

**Theorem: semi-decidability**

First-order logic (even restricted to only Horn clauses) is semi-decidable:
- If KB |= f, forward inference on complete inference rules will prove f in finite time.
- If KB \( \not\models f \), no algorithm can show this in finite time.

Resolution

**Recall:** First-order logic includes non-Horn clauses

\[ \forall x \text{Student}(x) \rightarrow \exists y \text{Knows}(x, y) \]

High-level strategy (same as in propositional logic):
- Convert all formulas to CNF
- Repeatedly apply resolution rule

Conversion to CNF

**Input:**

\[ \forall x (\forall y \text{Animal}(y) \rightarrow \text{Loves}(x, y)) \rightarrow \exists y \text{Loves}(y, x) \]

**Output:**

\[ (\text{Animal}(Y(x)) \lor \text{Loves}(Z(x), x)) \land (\neg \text{Loves}(x, Y(x)) \lor \text{Loves}(Z(x), x)) \]

**New to first-order logic:**
- All variables (e.g., \( x \)) have universal quantifiers by default
- Introduce **Skolem functions** (e.g., \( Y(x) \)) to represent existential quantified variables

• We can show that modus ponens is complete with respect to Horn clauses, which means that every true formula has an actual finite derivation.
• However, this doesn’t mean that we can just run modus ponens and be done with it, for first-order logic even restricted to Horn clauses is semi-decidable, which means that if a formula is entailed, then we will be able to derive it, but if it is not entailed, then we don’t even know when to stop the algorithm — quite troubling!
• With propositional logic, there were a finite number of propositional symbols, but now the number of atomic formulas can be infinite (the culprit is function symbols).
• Though we have hit a theoretical barrier, life goes on and we can still run modus ponens inference to get a one-sided answer. Next, we will move to working with full first-order logic.

• To go beyond Horn clauses, we will develop a single resolution rule which is sound and complete.
• The high-level strategy is the same as propositional logic: convert to CNF and apply resolution.

• Consider the logical formula corresponding to *Everyone who loves all animals is loved by someone.* The slide shows the desired output, which looks like a CNF formula in propositional logic, but there are two differences: there are variables (e.g., \( x \)) and functions of variables (e.g., \( Y(x) \)). The variables are assumed to be universally quantified over, and the functions are called **Skolem functions** and stand for a property of the variable.
Conversion to CNF (part 1)

Anyone who likes all animals is liked by someone.

Input:
\( \forall x (\forall y \text{Animal}(y) \rightarrow \text{Loves}(x, y)) \rightarrow \exists y \text{Loves}(y, x) \)

Eliminate implications (old):
\( \forall x (\neg \forall y \text{Animal}(y) \lor \text{Loves}(x, y)) \lor \exists y \text{Loves}(y, x) \)

Push \( \neg \) inwards, eliminate double negation (old):
\( \forall x (\exists y \text{Animal}(y) \land \neg \text{Loves}(x, y)) \lor \exists y \text{Loves}(y, x) \)

Standardize variables (new):
\( \forall x (\exists y \text{Animal}(y) \land \neg \text{Loves}(x, y)) \lor \exists z \text{Loves}(z, x) \)

Conversion to CNF (part 2)

\( \forall x (\exists y \text{Animal}(y) \land \neg \text{Loves}(x, y)) \lor \exists z \text{Loves}(z, x) \)

Replace existentially quantified variables with Skolem functions (new):
\( \forall x [\text{Animal}(Y(x)) \land \neg \text{Loves}(x, Y(x))] \lor \text{Loves}(Z(x), x) \)

Distribute \( \lor \) over \( \land \) (old):
\( \forall x [\text{Animal}(Y(x)) \lor \text{Loves}(Z(x), x)] \land [\neg \text{Loves}(x, Y(x)) \lor \text{Loves}(Z(x), x)] \)

Remove universal quantifiers (new):
\[ \text{Animal}(Y(x)) \lor \text{Loves}(Z(x), x) \land \neg \text{Loves}(x, Y(x)) \lor \text{Loves}(Z(x), x) \]

Resolution

Definition: resolution rule (first-order logic)
\[ f_1 \lor \cdots \lor f_n \lor p, \quad \neg q \lor g_1 \lor \cdots \lor g_m \]
\[ \text{Subst}[\theta, f_1 \lor \cdots \lor f_n \lor g_1 \lor \cdots \lor g_m] \]
\[ \text{where } \theta = \text{Unify}[p, q]. \]

Example: resolution

\[ \text{Animal}(Y(x)) \lor \text{Loves}(Z(x), x), \quad \neg \text{Loves}(u, v) \lor \text{Feeds}(u, v) \]
\[ \text{Animal}(Y(x)) \lor \text{Feeds}(Z(x), x) \]
\[ \text{Substitution: } \theta = \{ u/Z(x), v/x \}. \]
Summary

<table>
<thead>
<tr>
<th>Propositional logic</th>
<th>First-order logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>model checking</td>
<td>n/a</td>
</tr>
<tr>
<td>⇐ propositionalization</td>
<td></td>
</tr>
<tr>
<td>modus ponens (Horn clauses)</td>
<td>modus ponens++ (Horn clauses)</td>
</tr>
<tr>
<td>resolution (general)</td>
<td>resolution++ (general)</td>
</tr>
</tbody>
</table>

++: unification and substitution

Key idea: variables in first-order logic

Variables yield compact knowledge representations.

- To summarize, we have presented propositional logic and first-order logic. When there is a one-to-one mapping between constant symbols and objects, we can propositionalize, thereby converting first-order logic into propositional logic. This is needed if we want to use model checking to do inference.
- For inference based on syntactic derivations, there is a neat parallel between using modus ponens for Horn clauses and resolution for general formulas (after conversion to CNF). In the first-order logic case, things are more complex because we have to use unification and substitution to do matching of formulas.
- The main idea in first-order logic is the use of variables (not to be confused with the variables in variable-based models, which are mere propositional symbols from the point of view of logic), coupled with quantifiers.
- Propositional formulas allow us to express large complex sets of models compactly using a small piece of propositional syntax. Variables in first-order logic in essence takes this idea one more step forward, allowing us to effectively express large complex propositional formulas compactly using a small piece of first-order syntax.
- Note that variables in first-order logic are not same as the variables in variable-based models (CSPs). CSPs variables correspond to atomic formula and denote truth values. First-order logic variables denote objects.