Logic II

Lecture

Review

Propositional modus ponens

Propositional resolution

First order logic

First order modus ponens

First order resolution

- So far, we have used formulas, via semantics, to define sets of models. And all our reasoning on formulas has been through these models (e.g., reduction to satisfiability). Inference rules allow us to do reasoning on the formulas themselves without ever instantiating the models.
- This can be quite powerful. If you have a huge KB with lots of formulas and propositional symbols, sometimes you can draw a conclusion without instantiating the full model checking problem. This will be very important when we move to first-order logic, where the models can be infinite, and so model checking would be infeasible.

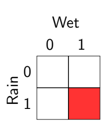
Review: ingredients of a logic

Syntax: defines a set of valid **formulas** (Formulas)

Example: Rain $\wedge$ Wet

Semantics: for each formula f, specify a set of **models** $\mathcal{M}(f)$ (assignments / configurations of the world)

Example:

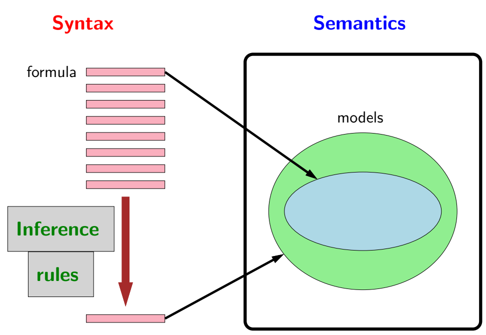


Inference rules: given KB, what new formulas f can be derived?

Example: from Rain $\wedge$ 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 $\mathcal{M}(f)$.

Propositional logic



Review: formulas

Propositional logic: any legal combination of symbols

(Rain $\wedge$ Snow) $\rightarrow$ (Traffic $\vee$ Peaceful) $\wedge$ Wet

- Whether a set of inference rules is complete depends on what the formulas are. Last time, we looked at propositional logic. In this lecture, we will also look at propositional logic restricted to Horn clauses (essentially formulas that look like $p_1 \wedge \cdots \wedge p_k \rightarrow q$), which intuitively can only derive positive information.

Review: inference algorithm

Inference algorithm:

KB —(repeatedly apply inference rules)→ f

Definition: modus ponens inference rule

$$\frac{p_1, \quad \cdots \quad, p_k, \quad (p_1 \wedge \cdots \wedge p_k) \rightarrow q}{q}$$

Desiderata: soundness and completeness

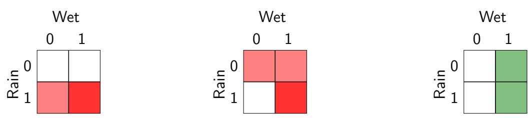
entailment (KB $\models f$) derivation (KB $\vdash f$)

- 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).
- 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.

Soundness: example

Is $\frac{\text{Rain}, \quad \text{Rain} \rightarrow \text{Wet}}{\text{Wet}}$ (Modus ponens) sound?

$\mathcal{M}(\text{Rain}) \quad \cap \quad \mathcal{M}(\text{Rain} \rightarrow \text{Wet}) \quad \subseteq? \quad \mathcal{M}(\text{Wet})$



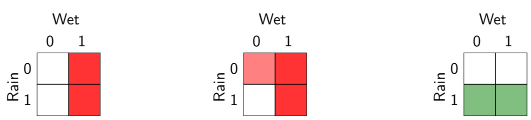
Sound!

- To check the soundness of a set of rules, it suffices to focus on one rule at a time.
- Take the modus ponens rule, for instance. We can derive Wet using modus ponens. To check entailment, we map all the formulas into semantics-land (the set of satisfiable models). Because the models of Wet is a superset of the intersection of models of Rain and Rain $\rightarrow$ Wet (remember that the models in the KB are an intersection of the models of each formula), we can conclude that Wet is also entailed. If we had other formulas in the KB, that would reduce both sides of $\subseteq$ by the same amount and won't affect the fact that the relation holds. Therefore, this rule is sound.
- Note, we use Wet and Rain to make the example more colorful, but this argument works for arbitrary propositional symbols.

Soundness: example

Is $\frac{\text{Wet}, \quad \text{Rain} \rightarrow \text{Wet}}{\text{Rain}}$ sound?

$\mathcal{M}(\text{Wet}) \quad \cap \quad \mathcal{M}(\text{Rain} \rightarrow \text{Wet}) \quad \subseteq? \quad \mathcal{M}(\text{Rain})$



Unsound!

- Here is another example: given Wet and Rain $\rightarrow$ Wet, can we infer Rain? To check it, we mechanically construct the models for the premises and conclusion. Here, the intersection of the models in the premise are not a subset, then the rule is unsound.
- Indeed, backward reasoning is faulty. Note that we can actually do a bit of backward reasoning using Bayesian networks, since we don't have to commit to 0 or 1 for the truth value.

Completeness: example

Recall completeness: inference rules derive all entailed formulas (f such that KB $\models f$)

Example: Modus ponens is incomplete

Setup:

KB $= \{\text{Rain}, \text{Rain} \vee \text{Snow} \rightarrow \text{Wet}\}$

$f = \text{Wet}$

Rules $= \{\frac{f, \quad f \rightarrow g}{g}\}$ (Modus ponens)

Semantically: KB $\models f$ (f is entailed).

Syntactically: KB $\not\vdash f$ (can't derive f).

Incomplete!

- Completeness is trickier, and here is a simple example that shows that modus ponens alone is not complete, since it can't derive Wet, when semantically, Wet is true!

Fixing completeness

Option 1: Restrict the allowed set of formulas

propositional logic

↓

propositional logic with only Horn clauses

Option 2: Use more powerful inference rules

Modus ponens

↓

resolution

- At this point, there are two ways to fix completeness. First, we can restrict the set of allowed formulas, making the water glass smaller in hopes that modus ponens will be able to fill that smaller glass.
- Second, we can use more powerful inference rules, pouring more vigorously into the same glass in hopes that this will be able to fill the glass; we'll look at one such rule, resolution, in the next lecture.

Lecture

Review

Propositional modus ponens

Propositional resolution

First order logic

First order modus ponens

First order resolution

Definite clauses

Definition: Definite clause

A **definite clause** has the following form:

$$(p_1 \wedge \cdots \wedge p_k) \rightarrow q$$

where $p_1, \dots, p_k, q$ are propositional symbols.

Intuition: if $p_1, \dots, p_k$ hold, then q holds.

Example: (Rain ∧ Snow) → Traffic

Example: Traffic

Non-example: ¬Traffic

Non-example: (Rain ∧ Snow) → (Traffic ∨ Peaceful)

- First we will choose to restrict the allowed set of formulas. Towards that end, let's define a **definite clause** as a formula that says, if a conjunction of propositional symbols holds, then some other propositional symbol q holds. Note that this is a formula, not to be confused with an inference rule.

Horn clauses

Definition: Horn clause

A **Horn clause** is either:

- a definite clause ($p_1 \wedge \cdots \wedge p_k \rightarrow q$)
- a goal clause ($p_1 \wedge \cdots \wedge p_k \rightarrow$ false)

Example (definite): (Rain $\wedge$ Snow) $\rightarrow$ Traffic

Example (goal): Traffic $\wedge$ Accident $\rightarrow$ false

equivalent: $\neg$(Traffic $\wedge$ Accident)

- A **Horn clause** is basically a definite clause, but includes another type of clause called a **goal clause**, which is the conjunction of a bunch of propositional symbols implying false. The form of the goal clause might seem a bit strange, but the way to interpret it is simply that it's the negation of the conjunction.

Modus ponens

Inference rule:

Definition: Modus ponens

$$\frac{p_1, \quad \cdots \quad, p_k, \quad (p_1 \wedge \cdots \wedge p_k) \rightarrow q}{q}$$

Example:

Example: Modus ponens

$$\frac{\text{Wet}, \quad \text{Weekday}, \quad \text{Wet} \wedge \text{Weekday} \rightarrow \text{Traffic}}{\text{Traffic}}$$

- Recall the Modus ponens rule from before. We simply have generalized it to arbitrary number of premises.

Completeness of modus ponens

Theorem: Modus ponens on Horn clauses

Modus ponens is **complete** with respect to Horn clauses:

- Suppose KB contains only Horn clauses and p is an entailed propositional symbol.
- Then applying modus ponens will derive p.

Upshot:

KB $\models p$ (entailment) is the same as KB $\vdash p$ (derivation)!

- There's a theorem that says that modus ponens is complete on Horn clauses. This means that any propositional symbol that is entailed can be derived by modus ponens too, provided that all the formulas in the KB are Horn clauses.
- We already proved that modus ponens is sound, and now we have that it is complete (for Horn clauses). The upshot of this is that entailment (a semantic notion, what we care about) and being able to derive a formula (a syntactic notion, what we do with inference) are equivalent!

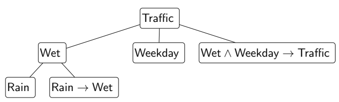
Example: Modus ponens

KB
Rain
Weekday
Rain → Wet
Wet ∧ Weekday → Traffic
Traffic ∧ Careless → Accident

Definition: Modus ponens

$$\frac{p_1, \quad \cdots \quad, p_k, \quad (p_1 \wedge \cdots \wedge p_k) \rightarrow q}{q}$$

Question: KB $\models$ Traffic $\Leftrightarrow$ KB $\vdash$ Traffic



- Let's see modus ponens on Horn clauses in action. Suppose we have the given KB consisting of only Horn clauses (in fact, these are all definite clauses), and we wish to ask whether the KB entails Traffic.
- We can construct a **derivation**, a tree where the root formula (e.g., Traffic) was derived using inference rules.
- The leaves are the original formulas in the KB, and each internal node corresponds to a formula which is produced by applying an inference rule (e.g., modus ponens) with the children as premises.
- If a symbol is used as the premise in two different rules, then it would have two parents, resulting in a DAG.

Lecture

Review

Propositional modus ponens

Propositional resolution

First order logic

First order modus ponens

First order resolution

Review: tradeoffs

Formulas allowed	**Inference rule**	**Complete?**
Propositional logic	modus ponens	no
Propositional logic (only Horn clauses)	modus ponens	yes
Propositional logic	**resolution**	yes

- 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.

Horn clauses and disjunction

Written with implication	Written with disjunction
$A \to C$	$\neg A \lor C$
$A \land B \to C$	$\neg A \lor \neg B \lor C$

- **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 $\neg A$

- 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.

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

$$\frac{\text{Rain} \lor \text{Snow}, \quad \neg\text{Snow} \lor \text{Traffic}}{\text{Rain} \lor \text{Traffic}}$$

Definition: resolution inference rule

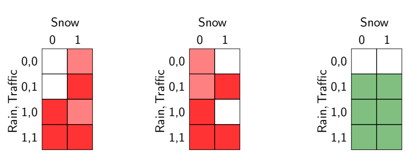
$$\frac{f_1 \lor \cdots \lor f_n \lor p, \quad \neg p \lor g_1 \lor \cdots \lor g_m}{f_1 \lor \cdots \lor f_n \lor g_1 \lor \cdots \lor g_m}$$

- 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, $f_1, \dots, f_n, g_1, \dots, g_m$ are arbitrary literals.

Soundness of resolution

$$\frac{\text{Rain} \lor \text{Snow}, \quad \neg\text{Snow} \lor \text{Traffic}}{\text{Rain} \lor \text{Traffic}} \text{ (resolution rule)}$$

$\mathcal{M}(\text{Rain} \lor \text{Snow}) \cap \mathcal{M}(\neg\text{Snow} \lor \text{Traffic}) \subseteq ?\ \mathcal{M}(\text{Rain} \lor \text{Traffic})$



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 $f \lor g$.

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: $(A \lor B \lor \neg C) \land (\neg B \lor D)$

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':

$$\mathcal{M}(f) = \mathcal{M}(f')$$

- 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. Given a CNF formula, we can toss each of its clauses into the knowledge base.
- But why can every formula be put in CNF?

Conversion to CNF: example

Initial formula:

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

Remove implication ($\rightarrow$):

$$\neg(\neg\text{Summer} \lor \text{Snow}) \lor \text{Bizzare}$$

Push negation ($\neg$) inwards (de Morgan):

$$(\neg\neg\text{Summer} \land \neg\text{Snow}) \lor \text{Bizzare}$$

Remove double negation:

$$(\text{Summer} \land \neg\text{Snow}) \lor \text{Bizzare}$$

Distribute $\lor$ over $\land$:

$$(\text{Summer} \lor \text{Bizzare}) \land (\neg\text{Snow} \lor \text{Bizzare})$$

- 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).

Conversion to CNF: general

Conversion rules:

- Eliminate $\leftrightarrow$: $\frac{f \leftrightarrow g}{(f \rightarrow g) \land (g \rightarrow f)}$
- Eliminate $\rightarrow$: $\frac{f \rightarrow g}{\neg f \lor g}$
- Move $\neg$ inwards: $\frac{\neg(f \land g)}{\neg f \lor \neg g}$
- Move $\neg$ inwards: $\frac{\neg(f \lor g)}{\neg f \land \neg g}$
- Eliminate double negation: $\frac{\neg\neg f}{f}$
- Distribute $\lor$ over $\land$: $\frac{f \lor (g \land h)}{(f \lor g) \land (f \lor h)}$

- 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: relationship between entailment and contradiction (basically "proof by contradiction")

KB $\models f$ ⟷ KB $\cup \{\neg f\}$ is unsatisfiable

Algorithm: resolution-based inference

- Add $\neg f$ into KB.
- Convert all formulas into **CNF**.
- Repeatedly apply **resolution** rule.
- Return entailment iff derive false.

- After we have converted all the formulas to CNF, we can repeatedly apply the resolution rule. But what is the final target?
- Recall that both testing for entailment and contradiction boil down to checking satisfiability. Resolution can be used to do this very thing. If we ever apply a resolution rule (e.g., to premises A and $\neg A$) and we derive false (which represents a contradiction), then the set of formulas in the knowledge base is unsatisfiable.
- If we are unable to derive false, that means the knowledge base is satisfiable because resolution is complete. However, unlike in model checking, we don't actually produce a concrete model that satisfies the KB.

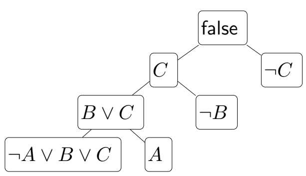
Resolution: example

KB$' = \{A \to (B \vee C), A, \neg B, \neg C\}$

Convert to CNF:

KB$' = \{\neg A \vee B \vee C, A, \neg B, \neg C\}$

Repeatedly apply **resolution** rule:



Conclusion: KB **entails** f

- Here's an example of taking a knowledge base, converting it into CNF, and applying resolution. In this case, we derive false, which means that the original knowledge base was unsatisfiable.

Time complexity

Definition: modus ponens inference rule

$$\frac{p_1, \quad \cdots \quad, p_k, \quad (p_1 \wedge \cdots \wedge p_k) \to q}{q}$$

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

Definition: resolution inference rule

$$\frac{f_1 \vee \cdots \vee f_n \vee p, \quad \neg p \vee g_1 \vee \cdots \vee g_m}{f_1 \vee \cdots \vee f_n \vee g_1 \vee \cdots \vee g_m}$$

- 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.

Summary

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

- 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.

Lecture

- Review
- Propositional modus ponens
- Propositional resolution
- **First order logic**
- First order modus ponens
- First order resolution

Limitations of propositional logic

Alice and Bob both know arithmetic.

AliceKnowsArithmetic $\wedge$ BobKnowsArithmetic

All students know arithmetic.

AliceIsStudent $\rightarrow$ AliceKnowsArithmetic

BobIsStudent $\rightarrow$ BobKnowsArithmetic

...

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

???

- 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.

Limitations of propositional logic

All students know arithmetic.

AliceIsStudent → AliceKnowsArithmetic

BobIsStudent → BobKnowsArithmetic

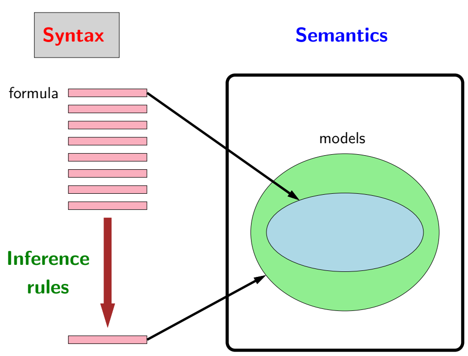
...

Propositional logic is very clunky. What's missing?

- Objects and predicates: 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 **predicates** on 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



- 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.

$$\text{Knows}(\text{alice}, \text{arithmetic}) \wedge \text{Knows}(\text{bob}, \text{arithmetic})$$

All students know arithmetic.

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

- Before formally defining things, let's look at two examples. First-order logic is basically propositional logic with a few more symbols.

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., $\text{Student}(x) \rightarrow \text{Knows}(x, \text{arithmetic})$)
- Quantifiers applied to formulas (e.g., $\forall x\, \text{Student}(x) \rightarrow \text{Knows}(x, \text{arithmetic})$)

- In propositional logic, everything was a formula (or a connective). In first-order logic, there are two types of beasts: terms and formulas. There are three types of terms: constant symbols (which refer to specific objects), variables (which refer to some unspecified object to be determined by quantifiers), and functions (which is a function applied to a set of arguments which are themselves terms).
- Given the terms, we can form atomic formulas, which are the analogue of propositional symbols, but with internal structure (e.g., terms).
- From this point, we can apply the same connectives on these atomic formulas, as we applied to propositional symbols in propositional logic. At this level, first-order logic looks very much like propositional logic.
- Finally, to make use of the fact that atomic formulas have internal structure, we have **quantifiers**, which are really the whole point of first-order logic!

Quantifiers

Universal quantification ($\forall$):

Think conjunction: $\forall x\, P(x)$ is like $P(A) \wedge P(B) \wedge \cdots$

Existential quantification ($\exists$):

Think disjunction: $\exists x\, P(x)$ is like $P(A) \vee P(B) \vee \cdots$

Some properties:

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

- There are two types of quantifiers: universal and existential. These are basically glorified ways of doing conjunction and disjunction, respectively.
- For crude intuition, we can think of conjunction and disjunction as very nice syntactic sugar, which can be rolled out into something that looks more like propositional logic. But quantifiers aren't just sugar, and it is important that they be compact, for sometimes the variable being quantified over can take on an infinite number of objects.
- That being said, the conjunction and disjunction intuition suffices for day-to-day guidance. For example, it should be intuitive that pushing the negation inside a universal quantifier (conjunction) turns it into a existential (disjunction), which was the case for propositional logic (by de Morgan's laws). Also, one cannot interchange universal and existential quantifiers any more than one can swap conjunction and disjunction in propositional logic.

Natural language quantifiers

Universal quantification ($\forall$):

Every student knows arithmetic.

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

Existential quantification ($\exists$):

Some student knows arithmetic.

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

Note the different connectives!

- Universal and existential quantifiers naturally correspond to the words *every* and *some*, respectively. But when converting English to formal logic, one must exercise caution.
- *Every* can be thought of as taking two arguments P and Q (e.g., *student* and *knows arithmetic*). The connective between P and Q is an implication (not conjunction, which is a common mistake). This makes sense because when we talk about every P, we are only restricting our attention to objects x for which $P(x)$ is true. Implication does exactly that.
- On the other hand, the connective for existential quantification is conjunction, because we're asking for an object x such that $P(x)$ and $Q(x)$ both hold.

Some examples of first-order logic

There is some course that every student has taken.

$$\exists y\, \mathsf{Course}(y) \wedge [\forall x\, \mathsf{Student}(x) \rightarrow \mathsf{Takes}(x, y)]$$

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

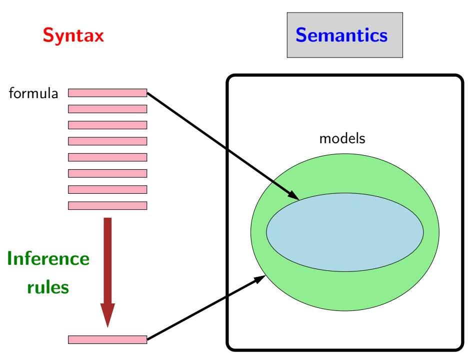
$$\forall x\, \mathsf{EvenInt}(x) \wedge \mathsf{Greater}(x, 2) \rightarrow \exists y\, \exists z\, \mathsf{Equals}(x, \mathsf{Sum}(y, z)) \wedge \mathsf{Prime}(y) \wedge \mathsf{Prime}(z)$$

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

$$\forall x\, \forall y\, \forall z\, (\mathsf{Student}(x) \wedge \mathsf{Takes}(x, y) \wedge \mathsf{Course}(y) \wedge \mathsf{Covers}(y, z)) \rightarrow \mathsf{Knows}(x, z)$$

- 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.

First-order logic



- 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.

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.

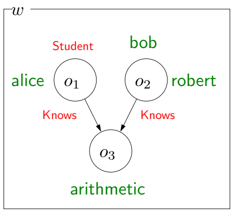
$$w = \{\mathsf{AliceKnowsArithmetic} : 1, \mathsf{BobKnowsArithmetic} : 0\}$$

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 predicates, a model w can be represented as a directed graph:



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

- A better way to think about a first-order model is that there are a number of objects in the world ($o_1, o_2, \dots$); think of these as nodes in a graph. Then we have predicates between these objects. Predicates that take two arguments can be visualized as labeled edges between objects. Predicates that take one argument can be visualized as node labels (but these are not so important).
- So far, the objects are unnamed. We can access individual objects directly using constant symbols, which are labels on the nodes.

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

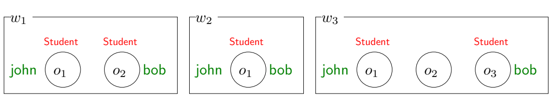
$w(\text{Knows}) = \{(o_1, o_3), (o_2, o_3), \dots\}$

- Formally, a first-order model w maps constant symbols to objects and predicate symbols to tuples of objects (2 for binary predicates).

A restriction on models

John and Bob are students.

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



- 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 ⟷ object

- Note that by default, two constant symbols can refer to the same object, and there can be objects which no constant symbols refer to. This can make life somewhat confusing. Fortunately, there are two assumptions that people sometimes make to simplify things.
- The unique names assumption says that there's at most one way to refer to an object via a constant symbol. Domain closure says there's at least one. Together, they imply that there is a one-to-one relationship between constant symbols in syntax-land and objects in semantics-land.

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

$\text{Student}(\text{alice}) \wedge \text{Student}(\text{bob})$
$\forall x \, \text{Student}(x) \rightarrow \text{Person}(x)$
$\exists x \, \text{Student}(x) \wedge \text{Creative}(x)$

Knowledge base in propositional logic

$\text{Studentalice} \wedge \text{Studentbob}$
$(\text{Studentalice} \rightarrow \text{Personalice}) \wedge (\text{Studentbob} \rightarrow \text{Personbob})$
$(\text{Studentalice} \wedge \text{Creativealice}) \vee (\text{Studentbob} \wedge \text{Creativebob})$

Point: use any inference algorithm for propositional logic!

- If a one-to-one mapping really exists, then we can **propositionalize** all our formulas, which basically unrolls all the quantifiers into explicit conjunctions and disjunctions.
- The upshot of this conversion, is that we're back to propositional logic, and we know how to do inference in propositional logic (either using model checking or by applying inference rules). Of course, propositionalization could be quite expensive and not the most efficient thing to do.

Lecture

Review

Propositional modus ponens

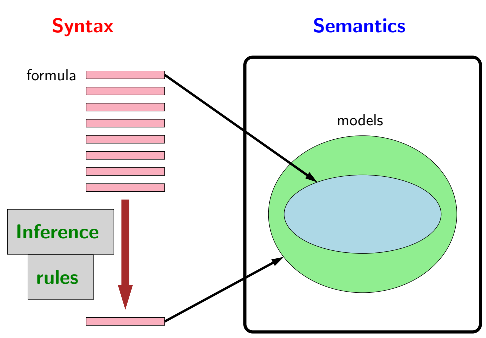
Propositional resolution

First order logic

First order modus ponens

First order resolution

First-order logic



- Now we look at inference rules which can make first-order inference much more efficient. The key is to do everything implicitly and avoid propositionalization; again the whole spirit of logic is to do things compactly and implicitly.

Definite clauses

$$\forall x\, \forall y\, \forall z\, (\text{Takes}(x, y) \wedge \text{Covers}(y, z)) \rightarrow \text{Knows}(x, z)$$

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

Definition: definite clause (first-order logic)

A definite clause has the following form:

$$\forall x_1 \cdots \forall x_n\, (a_1 \wedge \cdots \wedge a_k) \rightarrow b$$

for variables $x_1, \ldots, x_n$ and atomic formulas $a_1, \ldots, a_k, b$ (which contain those variables).

- Like our development of inference in propositional logic, we will first talk about first-order logic restricted to Horn clauses, in which case a first-order version of modus ponens will be sound and complete. After that, we'll see how resolution allows to handle all of first-order logic.
- We start by generalizing definite clauses from propositional logic to first-order logic. The only difference is that we now have universal quantifiers sitting at the beginning of the definite clause. This makes sense since universal quantification is associated with implication, and one can check that if one propositionalizes a first-order definite clause, one obtains a set (conjunction) of multiple propositional definite clauses.

Modus ponens (first attempt)

Definition: modus ponens (first-order logic)

$$\frac{a_1, \ldots, a_k \quad \forall x_1 \cdots \forall x_n (a_1 \wedge \cdots \wedge a_k) \rightarrow b}{b}$$

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

- If we try to write down the modus ponens rule, we would fail.
- As a simple example, suppose we are given $P(\text{alice})$ and $\forall x\, P(x) \rightarrow Q(x)$. We would naturally want to derive $Q(\text{alice})$. But notice that we can't apply modus ponens because $P(\text{alice})$ and $P(x)$ don't match!
- Recall that we're in syntax-land, which means that these formulas are just symbols. Inference rules don't have access to the semantics of the constants and variables — it is just a pattern matcher. So we have to be very methodical.
- To develop a mechanism to match variables and constants, we will introduce two concepts, substitution and unification for this purpose.

Substitution

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

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

Definition: Substitution

A substitution θ is a mapping from variables to terms.

$\text{Subst}[\theta, f]$ returns the result of performing substitution θ on f.

- The first step is substitution, which applies a search-and-replace operation on a formula or term.
- We won't define $\text{Subst}[\theta, f]$ formally, but from the examples, it should be clear what Subst does.
- Technical note: if θ contains variable substitutions x/alice we only apply the substitution to the free variables in f, which are the variables not bound by quantification (e.g., x in $\exists y, P(x, y)$). Later, we'll see how CNF formulas allow us to remove all the quantifiers.

Unification

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

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

Unify[Knows(alice, y), Knows(bob, z)] = fail

Unify[Knows(alice, y), Knows(x, $F(x)$)] = {x/alice, y/F(alice)}

Definition: Unification

Unification takes two formulas f and g and returns a substitution θ which is the most general unifier:

Unify[f, g] = θ such that Subst[θ, f] = Subst[θ, g]

or "fail" if no such θ exists.

- Substitution can be used to make two formulas identical, and unification is the way to find the least committal substitution we can find to achieve this.
- Unification, like substitution, can be implemented recursively. The implementation details are not the most exciting, but it's useful to get some intuition from the examples.

Modus ponens

Definition: modus ponens (first-order logic)

$$\frac{a'_1, \dots, a'_k \quad \forall x_1 \cdots \forall x_n (a_1 \wedge \cdots \wedge a_k) \rightarrow b}{b'}$$

Get most general unifier θ on premises:

- $\theta = \text{Unify}[a'_1 \wedge \cdots \wedge a'_k, a_1 \wedge \cdots \wedge a_k]$

Apply θ to conclusion:

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

- Having defined substitution and unification, we are in position to finally define the modus ponens rule for first-order logic. Instead of performing a exact match, we instead perform a unification, which generates a substitution θ. Using θ, we can generate the conclusion b' on the fly.
- Note the significance here: the rule $a_1 \wedge \cdots \wedge a_k \rightarrow b$ can be used in a myriad ways, but Unify identifies the appropriate substitution, so that it can be applied to the conclusion.

Modus ponens example

Example: modus ponens in first-order logic

Premises:

Takes(alice, cs221)

Covers(cs221, mdp)

$\forall x \, \forall y \, \forall z$ Takes(x, y) $\wedge$ Covers(y, z) $\rightarrow$ Knows(x, z)

Conclusion:

θ = {x/alice, y/cs221, z/mdp}

Derive Knows(alice, mdp)

- 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).

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...

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

- 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 $\models f$, 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.

- 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.

Lecture

Review

Propositional modus ponens

Propositional resolution

First order logic

First order modus ponens

First order resolution

First-order resolution

Resolution

Recall: First-order logic includes non-Horn clauses

$$\forall x\, \mathsf{Student}(x) \to \exists y\, \mathsf{Knows}(x, y)$$

High-level strategy (same as in propositional logic):

- Convert all formulas to CNF
- Repeatedly apply resolution rule

- 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.

Conversion to CNF

Input:

$$\forall x\, (\forall y\, \mathsf{Animal}(y) \to \mathsf{Loves}(x, y)) \to \exists y\, \mathsf{Loves}(y, x)$$

Output:

$$(\mathsf{Animal}(Y(x)) \lor \mathsf{Loves}(Z(x), x)) \land (\neg \mathsf{Loves}(x, Y(x)) \lor \mathsf{Loves}(Z(x), x))$$

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

- 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 loves all animals is loved 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):

$\forall x\,\neg(\forall y\,\neg\text{Animal}(y) \vee \text{Loves}(x,y)) \vee \exists y\,\text{Loves}(y,x)$

Push $\neg$ inwards, eliminate double negation (old):

$\forall x\,(\exists y\,\text{Animal}(y) \wedge \neg\text{Loves}(x,y)) \vee \exists y\,\text{Loves}(y,x)$

Standardize variables (**new**):

$\forall x\,(\exists y\,\text{Animal}(y) \wedge \neg\text{Loves}(x,y)) \vee \exists z\,\text{Loves}(z,x)$

- We start by eliminating implications, pushing negation inside, and eliminating double negation, which is all old.
- The first thing new to first-order logic is standardization of variables. Note that in $\exists x\,P(x) \wedge \exists x\,Q(x)$, there are two instances of x whose scopes don't overlap. To make this clearer, we will convert this into $\exists x\,P(x) \wedge \exists y\,Q(y)$. This sets the stage for when we will drop the quantifiers on the variables.

Conversion to CNF (part 2)

$\forall x\,(\exists y\,\text{Animal}(y) \wedge \neg\text{Loves}(x,y)) \vee \exists z\,\text{Loves}(z,x)$

Replace existentially quantified variables with Skolem functions (**new**):

$\forall x\,[\text{Animal}(Y(x)) \wedge \neg\text{Loves}(x,Y(x))] \vee \text{Loves}(Z(x),x)$

Distribute $\vee$ over $\wedge$ (old):

$\forall x\,[\text{Animal}(Y(x)) \vee \text{Loves}(Z(x),x)] \wedge [\neg\text{Loves}(x,Y(x)) \vee \text{Loves}(Z(x),x)]$

Remove universal quantifiers (**new**):

$[\text{Animal}(Y(x)) \vee \text{Loves}(Z(x),x)] \wedge [\neg\text{Loves}(x,Y(x)) \vee \text{Loves}(Z(x),x)]$

- The next step is to remove existential variables by replacing them with Skolem functions. This is perhaps the most non-trivial part of the process. Consider the formula: $\forall x\,\exists y\,P(x,y)$. Here, y is existentially quantified and depends on x. So we can mark this dependence explicitly by setting $y = Y(x)$. Then the formula becomes $\forall x\,P(x,Y(x))$. You can even think of the function Y as being existentially quantified over outside the $\forall x$.
- Next, we distribute disjunction over conjunction as before.
- Finally, we simply drop all universal quantifiers. Because those are the only quantifiers left, there is no ambiguity.
- The final CNF formula can be difficult to interpret, but we can be assured that the final formula captures exactly the same information as the original formula.

Resolution

Definition: resolution rule (first-order logic)

$$\frac{f_1 \vee \cdots \vee f_n \vee p, \quad \neg q \vee g_1 \vee \cdots \vee g_m}{\text{Subst}[\theta, f_1 \vee \cdots \vee f_n \vee g_1 \vee \cdots \vee g_m]}$$

where $\theta = \text{Unify}[p,q]$.

Example: resolution

$$\frac{\text{Animal}(Y(x)) \vee \text{Loves}(Z(x),x), \quad \neg\text{Loves}(u,v) \vee \text{Feeds}(u,v)}{\text{Animal}(Y(x)) \vee \text{Feeds}(Z(x),x)}$$

Substitution: $\theta = \{u/Z(x), v/x\}$.

- After convering all formulas to CNF, then we can apply the resolution rule, which is generalized to first-order logic. This means that instead of doing exact matching of a literal p, we unify atomic formulas p and q, and then apply the resulting substitution θ on the conclusion.

Summary

Propositional logic	**First-order logic**
model checking	n/a
	⇐ propositionalization
modus ponens (Horn clauses)	modus ponens++ (Horn clauses)
resolution (general)	resolution++ (general)

++: 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.