



Lecture 13.1: Logic II



Review: ingredients of a logic

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

Example: $\text{Rain} \wedge \text{Wet}$

Semantics: for each formula, specify a set of **models** (assignments / configurations of the world)

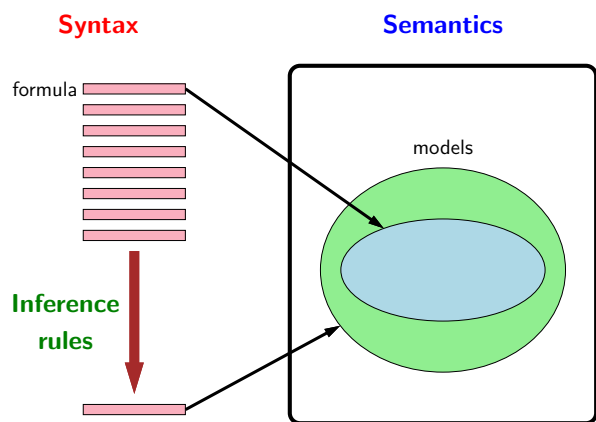
Example:

		Wet	
		0	1
Rain	0		
	1		

Inference rules: given f , what new formulas g can be added that are guaranteed to follow ($\frac{f}{g}$)?

Example: from $\text{Rain} \wedge \text{Wet}$, derive Rain

Review: schema



Review: inference task

Inference task

Input:

Knowledge base KB (e.g., $\{\text{Rain}, \text{Rain} \rightarrow \text{Wet}\}$)

Query formula f (e.g., Wet)

Output:

Whether KB entails f ($\text{KB} \models f$)? (e.g., yes)

$(\text{KB} \models f \iff \mathcal{M}(f) \supseteq \mathcal{M}(\text{KB}))$

- 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 $\mathcal{M}(f)$.

- 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:

KB $\xrightarrow{\text{(repeatedly apply inference rules)}} f$



Definition: modus ponens inference rule

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

Desiderata: soundness and completeness



- 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 and completeness

The truth, the whole truth, and nothing but the truth.

- **Soundness:** nothing but the truth
- **Completeness:** whole truth

- A slogan to keep in mind is the oath given in a sworn testimony.

Soundness: example

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

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

		Wet	
		0	1
Rain	0		
	1		

		Wet	
		0	1
Rain	0		
	1		

		Wet	
		0	1
Rain	0		
	1		

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

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

		Wet	
		0	1
Rain	0		
	1		

		Wet	
		0	1
Rain	0		
	1		

		Wet	
		0	1
Rain	0		
	1		

Unsound!

- Here is another example: given Wet and $\text{Rain} \rightarrow \text{Wet}$, 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 $\text{KB} \models f$)



Example: Modus ponens is incomplete

Setup:

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

$f = \text{Wet}$

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

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

Syntactically: $\text{KB} \not\vdash f$ (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!

Completeness of Modus ponens

Modus ponens: not complete for all formulas

- Is complete for Horn clauses

Next: more powerful inference rules

Modus ponens



resolution

- What do we do about the fact that Modus ponens is not complete? First, we could decide that we only care about *some* types of formulas. You can think of this as making the water glass smaller in hopes that modus ponens will be able to fill that smaller glass. It turns out that modus ponens is complete for a subset of formulas called Horn clauses. We won't discuss exactly what these are in this class (I will post some supplemental slides if you're curious).
- 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.

Approaches

Formulas allowed Inference rule Complete?

Propositional logic (only Horn clauses) modus ponens yes

Propositional logic modus ponens no

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.



Roadmap

Resolution in propositional logic

First-order logic

Modus ponens with disjunction

Written with implication

$$A \rightarrow C$$

$$A \wedge B \rightarrow C$$

Written with disjunction

$$\neg A \vee C$$

$$\neg A \vee \neg B \vee C$$

- **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):

$$\frac{A, \neg A \vee C}{C}$$

- Intuition: cancel out A and $\neg A$

- We can equivalently write implication using negation and disjunction. The types of implications that modus ponens works with 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 $\{A, \neg A \vee C\} \models C$; this is equivalent to: if $A \wedge (\neg A \vee C)$ 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:

$$\neg A \vee B \vee \neg C \vee D \vee \neg E \vee F$$



Example: resolution inference rule

$$\frac{\text{Rain} \vee \text{Snow}, \neg \text{Snow} \vee \text{Traffic}}{\text{Rain} \vee \text{Traffic}}$$



Definition: resolution inference rule

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

- 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

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

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

		Snow	
		0	1
Rain, Traffic	0,0		
	0,1		
	1,0		
	1,1		

		Snow	
		0	1
Rain, Traffic	0,0		
	0,1		
	1,0		
	1,1		

		Snow	
		0	1
Rain, Traffic	0,0		
	0,1		
	1,0		
	1,1		

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 $f \vee g$.

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: $(A \vee B \vee \neg C) \wedge (\neg B \vee D)$

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):

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

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

$$(\neg \neg \text{Summer} \wedge \neg \text{Snow}) \vee \text{Bizzare}$$

Remove double negation:

$$(\text{Summer} \wedge \neg \text{Snow}) \vee \text{Bizzare}$$

Distribute \vee over \wedge :

$$(\text{Summer} \vee \text{Bizzare}) \wedge (\neg \text{Snow} \vee \text{Bizzare})$$

- 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).

- 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.

- 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 is clearly 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.

Conversion to CNF: general

Conversion rules:

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

CS221 / Summer 2019 / Jia

31

Resolution algorithm

Recall: entailment and contradiction \Leftrightarrow satisfiability

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

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



Algorithm: resolution-based inference

- Convert all formulas into **CNF**.
- Repeatedly apply **resolution** rule.
- Return unsatisfiable iff derive false.

CS221 / Summer 2019 / Jia

33

Resolution: example

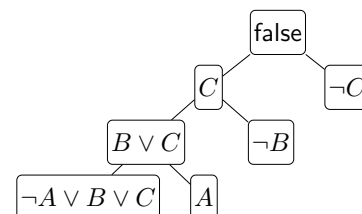
Knowledge base (is it satisfiable?):

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

Convert to CNF:

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

Repeatedly apply **resolution** rule:



Unsatisfiable!

CS221 / Summer 2019 / Jia

35

- 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.

- 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.

Time complexity



Definition: modus ponens inference rule

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

- Each rule application adds clause with **one** propositional symbol
⇒ linear time



Definition: resolution inference rule

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

- Each rule application adds clause with **many** propositional symbols
⇒ 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.



Roadmap

Resolution in propositional logic

First-order logic

Limitations of propositional logic

Alice and Bob both know arithmetic.

$\text{AliceKnowsArithmetic} \wedge \text{BobKnowsArithmetic}$

All students know arithmetic.

$\text{AliceIsStudent} \rightarrow \text{AliceKnowsArithmetic}$

$\text{BobIsStudent} \rightarrow \text{BobKnowsArithmetic}$

...

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.

$\text{AliceIsStudent} \rightarrow \text{AliceKnowsArithmetic}$

$\text{BobIsStudent} \rightarrow \text{BobKnowsArithmetic}$

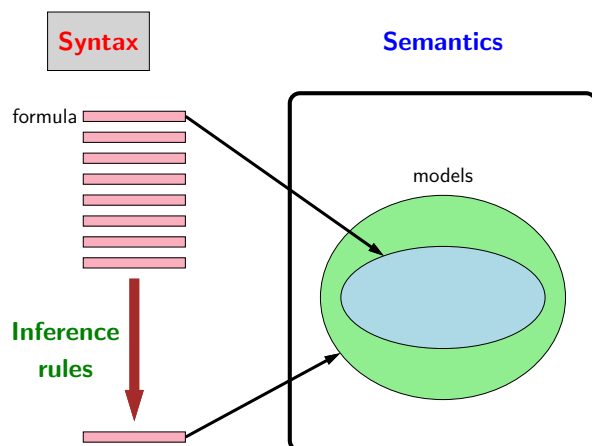
...

Propositional logic is very clunky. What's missing?

- **Objects and relations:** propositions (e.g., $\text{AliceKnowsArithmetic}$) 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



- 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.

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

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

Quantifiers

Universal quantification (\forall):

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

Existential quantification (\exists):

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

Some properties:

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

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

- 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!

- 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 an 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.

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

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

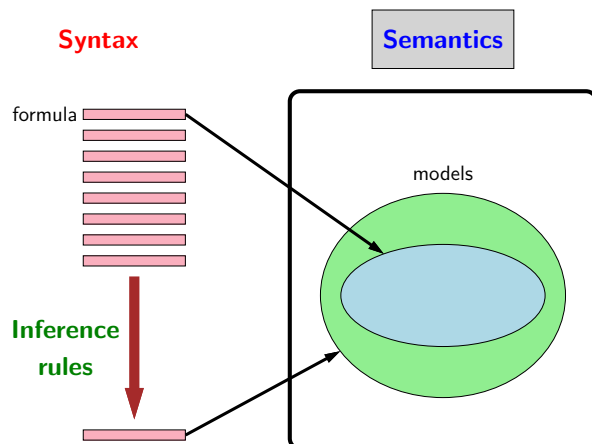
$\forall x \text{ EvenInt}(x) \wedge \text{Greater}(x, 2) \rightarrow \exists y \exists z \text{ Equals}(x, \text{Sum}(y, z)) \wedge \text{Prime}(y) \wedge \text{Prime}(z)$

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

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

- 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.

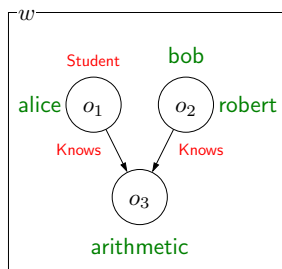
$$w = \{\text{AliceKnowsArithmetic} : 1, \text{BobKnowsArithmetic} : 0\}$$

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 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

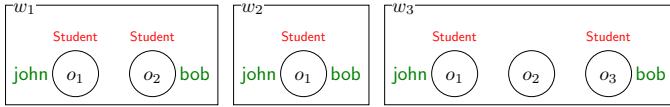
$$w(\text{Knows}) = \{(o_1, o_3), (o_2, o_3), \dots\}$$

- 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 \longleftrightarrow 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

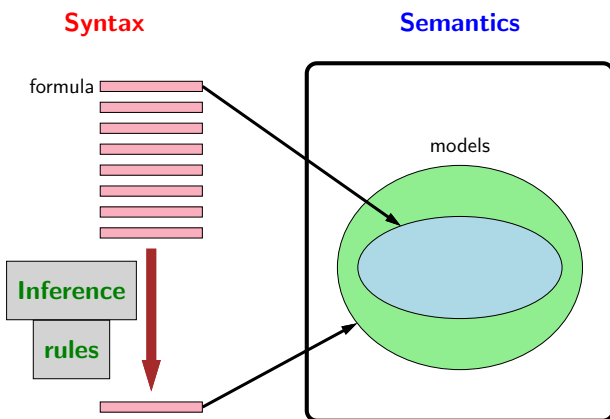
$\text{Student}(\text{alice}) \wedge \text{Student}(\text{bob})$
 $\forall x \text{ Student}(x) \rightarrow \text{Person}(x)$
 $\exists x \text{ Student}(x) \wedge \text{Creative}(x)$

Knowledge base in propositional logic

$\text{Student}(\text{alice}) \wedge \text{Student}(\text{bob})$
 $(\text{Student}(\text{alice}) \rightarrow \text{Person}(\text{alice})) \wedge (\text{Student}(\text{bob}) \rightarrow \text{Person}(\text{bob}))$
 $(\text{Student}(\text{alice}) \wedge \text{Creative}(\text{alice})) \vee (\text{Student}(\text{bob}) \wedge \text{Creative}(\text{bob}))$

Point: use any inference algorithm for propositional logic!

First-order logic



- 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.

- 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.

- 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.

Modus ponens example



Example: modus ponens in first-order logic

Premises:

Takes(alice, cs221)

Covers(cs221, mdp)

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

Conclusion:

$\theta = \{x/\text{alice}, y/\text{cs221}, z/\text{mdp}\}$

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).

Modus ponens



Definition: modus ponens (first-order logic)

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

Unification: Get most general unifier θ on premises

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

Substitution: 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 $a_1 \wedge \dots \wedge a_k \rightarrow b$ can be used in a myriad ways, but Unify identifies the appropriate substitution, so that it can be applied to the conclusion.

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...

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

- 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 (recognizable)**.

- 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.

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

- 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:

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

Output:

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

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

- 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.



Summary

Propositional logic

model checking

⇐ propositionalization

modus ponens
(Horn clauses)

resolution
(general)

++: unification and substitution



Key idea: variables in first-order logic

Variables yield compact knowledge representations.

First-order logic

n/a

modus ponens++
(Horn clauses)

resolution++
(general)

- 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.