Logic

Question

think and share

If $X_1 + X_2 = 10$ and $X_1 - X_2 = 4$, what is X_1?

- Think about how you solved this problem. You could treat it as a CSP with variables X_1 and X_2, and search through the set of candidate solutions, checking the constraints.
- However, more likely, you just added the two equations, divided both sides by 2 to easily find out that $X_1 = 7$. This is the power of **logical inference**, where we apply a set of truth-preserving rules to arrive at the answer. This is in contrast to what is called **model checking** (for reasons that will become clear), which tries to directly find assignments.
- We'll see that logical inference allows you to perform very powerful manipulations in a very compact way. This allows us to vastly increase the representational power of our models.

Course plan

Search problems | Constraint satisfaction problems
Markov decision processes | Markov networks
Adversarial games | Bayesian networks

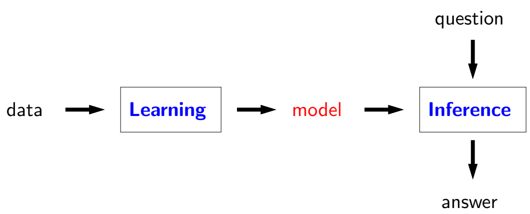
Reflex → **States** → **Variables** → $\boxed{\textbf{Logic}}$

Low-level → High-level

Machine learning

- We are at the last stage of our journey through the AI topics of this course: logic. Before launching in, let's take a moment to reflect.

Taking a step back



Examples: search problems, MDPs, games, CSPs, Bayesian networks

- For each topic (e.g., MDPs) that we've studied, we followed the modeling-inference-learning paradigm: We take some data, feed it into a learning algorithm to produce a model with tuned parameters. Then we take this model and use it to perform inference (turning questions into answers).
- For search problems, the question is "what is the minimum cost path?" Inference algorithms such as DFS, UCS or A* produced the minimum cost path. Learning algorithms such as the structured Perceptron filled in the action costs based on data (minimum cost paths).
- For MDPs and games, the question is "what is the maximum value policy?" Inference algorithms such as value iteration or minimax produced this. Learning algorithms such as Q-learning or TD learning allow you to work when we don't know the transitions and rewards.
- For CSPs, the question is "what is the maximum weight assignment?" Inference algorithms such as backtracking search, beam search, or variable elimination find such an assignment. We did not discuss learning algorithms here, but something similar to the structured Perceptron works.
- For Bayesian networks, the question is "what is the probability of a query given evidence?" Inference algorithms such as Gibbs sampling and particle filtering compute these probabilistic inference queries. Learning: if we don't know the local conditional distributions, we can learn them using maximum likelihood.
- We can think of learning as induction, where we need to generalize, and inference as deduction, where it's about computing the best predicted answer under the model.

Modeling paradigms

State-based models: search problems, MDPs, games

Applications: route finding, game playing, etc.
Think in terms of **states, actions, and costs**

Variable-based models: CSPs, Bayesian networks

Applications: scheduling, tracking, medical diagnosis, etc.
Think in terms of **variables and factors**

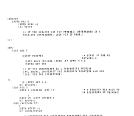
Logic-based models: propositional logic, first-order logic

Applications: theorem proving, verification, reasoning
Think in terms of **logical formulas and inference rules**

- Each topic corresponded to a modeling paradigm. The way the modeling paradigm is set up influences the way we approach a problem.
- In state-based models, we thought about inference as finding minimum cost paths in a graph. This leads us to think in terms of states, actions, and costs.
- In variable-based models, we thought about inference as finding maximum weight assignments or computing conditional probabilities. There we thought about variables and factors.
- Now, we will talk about logic-based models, where inference is applying a set of rules. For these models, we will think in terms of logical formulas and inference rules.

A historical note

- Logic was dominant paradigm in AI before 1990s



- Problem 1: deterministic, didn't handle **uncertainty** (probability addresses this)
- Problem 2: rule-based, didn't allow fine tuning from **data** (machine learning addresses this)
- Strength: provides **expressiveness** in a compact way

- Historically, in AI, logic was the dominant paradigm before the 1990s, but this tradition fell out of favor with the rise of probability and machine learning.
- There were two reasons for this: First, logic as an inference mechanism was brittle and did not handle uncertainty, whereas probability offered a coherent framework for dealing with uncertainty.
- Second, people built rule-based systems which were tedious and did not scale up, whereas machine learning automated much of the fine-tuning of a system by using data.
- However, there is one strength of logic which has not quite yet been recouped by existing probability and machine learning methods, and that is the expressivity of the model.

Motivation: smart personal assistant

- How can we motivate logic-based models? We will take a little bit of a detour and think about an AI grand challenge: building smart personal assistants.
- Today, we have systems like Apple's Siri, Microsoft's Cortana, Amazon's Alexa, and Google Assistant.

Motivation: smart personal assistant

Tell information →  ← **Ask** questions

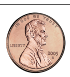
Use natural language!

[demo: `python nli.py`]

Need to:

- Digest **heterogenous** information
- Reason **deeply** with that information

- We would like to have more intelligent assistants such as Data from Star Trek. What is the functionality that's missing in between?
- At an abstract level, one fundamental thing a good personal assistant should be able to do is to take in information from people and be able to answer questions that require drawing inferences from the facts.
- In some sense, telling the system information is like machine learning, but it feels like a very different form of learning than seeing 10M images and their labels or 10M sentences and their translations. The type of information we get here is both more heterogenous, more abstract, and the expectation is that we process it more deeply (we don't want to have to tell our personal assistant 100 times that we prefer morning meetings).
- And how do we interact with our personal assistants? Let's use natural language, the very tool that was built for communication!



Natural language

Example:

- A **dime** is better than a **nickel**.
- A **nickel** is better than a **penny**.
- Therefore, a **dime** is better than a **penny**.

Example:

- A **penny** is better than **nothing**.
- **Nothing** is better than **world peace**.
- Therefore, a **penny** is better than **world peace**???

Natural language is slippery...

- But natural language is tricky, because it is replete with ambiguities and vagueness. And drawing inferences using natural languages can be quite slippery. Of course, some concepts are genuinely vague and slippery, and natural language is as good as it gets, but that still leaves open the question of how a computer would handle those cases.

Language

Language *is a mechanism for expression.*

Natural languages (informal):

English: *Two divides even numbers.*
German: *Zwei dividiert gerade Zahlen.*

Programming languages (formal):

Python: `def even(x): return x % 2 == 0`
C++: `bool even(int x) { return x % 2 == 0; }`

Logical languages (formal):

First-order-logic: $\forall x.\text{Even}(x) \rightarrow \text{Divides}(x, 2)$

- Let's think about language a bit deeply. What does it really buy you? Primarily, language is this wonderful human creation that allows us to express and communicate complex ideas and thoughts.
- We have mostly been talking about natural languages such as English and German. But as you all know, there are programming languages as well, which allow one to express computation formally so that a computer can understand it.
- This lecture is mostly about logical languages such as propositional logic and first-order logic. These are formal languages, but are a more suitable way of capturing declarative knowledge rather than concrete procedures, and are better connected with natural language.

Two goals of a logic language

- **Represent** knowledge about the world

- **Reason** with that knowledge

- Some of you already know about logic, but it's important to keep the AI goal in mind: We want to use it to represent knowledge, and we want to be able to reason (or do inference) with that knowledge.
- Finally, we need to keep in mind that our goal is to get computers to use logic automatically, not for you to do it. This means that we need to think very mechanistically.

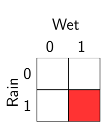
Ingredients of a logic

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

Example: Rain $\wedge$ Wet

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

Example:



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

Example: from Rain $\wedge$ Wet, derive Rain

- The **syntax** defines a set of valid formulas, which are things which are grammatical to say in the language.
- **Semantics** usually doesn't receive much attention if you have a casual exposure to logic, but this is really the important piece that makes logic rigorous. Formally, semantics specifies the meaning of a formula, which in our setting is a set of configurations of the world in which the formula holds. This is what we care about in the end.
- But in order to get there, it's helpful to operate directly on the syntax using a set of **inference rules**. For example, if I tell you that it's raining and wet, then you should be able to conclude that it is also raining (obviously) without even explicitly mentioning semantics. Most of the time when people do logic casually, they are really just applying inference rules.

Syntax versus semantics

Syntax: what are valid expressions in the language?

Semantics: what do these expressions mean?

Different syntax, same semantics (5):

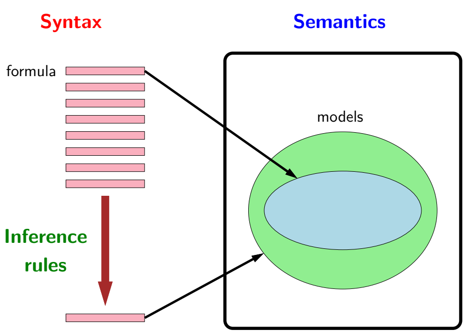
$$\texttt{2 + 3} \quad \Leftrightarrow \quad \texttt{3 + 2}$$

Same syntax, different semantics (1 versus 1.5):

$$\texttt{3 / 2} \text{ (Python 2.7)} \quad \not\Leftrightarrow \quad \texttt{3 / 2} \text{ (Python 3)}$$

- Just to hammer in the point that syntax and semantics are different, consider two examples from programming languages.
- First, the formula 2 + 3 and 3 + 2 are superficially different (a syntactic notion), but they have the same semantics (5).
- Second, the formula 3 / 2 means something different depending on which language. In Python 2.7, the semantics is 1 (integer division), and in Python 3 the semantics is 1.5 (floating point division).

Propositional logic



Logics

- **Propositional logic with only Horn clauses**
- **Propositional logic**
- Modal logic
- **First-order logic with only Horn clauses**
- **First-order logic**
- Second-order logic
- ...

Key idea: tradeoff

Balance **expressivity** and **computational efficiency**.

- There are many different logical languages, just like there are programming languages. Whereas most programming languages have the expressive power (all Turing complete), logical languages exhibit a larger spectrum of expressivity.
- The bolded items are the ones we will discuss in this class.

Roadmap

Modeling	**Inference**
Propositional Logic Syntax	Inference Rules
Propositional Logic Semantics	Propositional modus ponens
First-order Logic	Propositional resolution
	First-order modus ponens
	First-order resolution

- Here are the rest of the modules under the logic unit.
- We will start by talking about the core elemenst of logics: syntax, semantics, and inference rules. We will start by defining syntax and semantics for propositional logic.
- We will then discuss a set of inference rules for propostional logic including modus ponens and resolution. We will discuss soundness and correctness of these inference algorithms.
- We then describe a more expressive logic, i.e., first order logic. We will go over its syntax and semantics, and then extend the notions of modus ponens and resolution to first order logic using unification and substitution.

Lecture

Overview

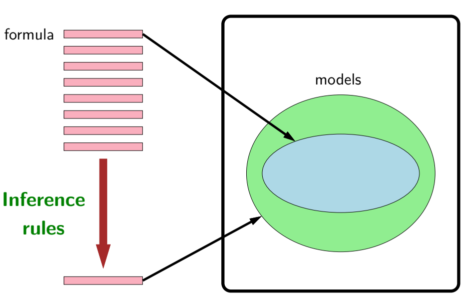
Propositional logic syntax

Propositional logic semantics

Inference rules

Propositional logic

Syntax **Semantics**



- We begin with the syntax of propositional logic: what are the allowable formulas?

Syntax of propositional logic

Propositional symbols (atomic formulas); A, B, C:

Logical connectives: $\neg, \wedge, \vee, \rightarrow, \leftrightarrow$

Build up formulas recursively—if f and g are formulas, so are the following:

- Negation: $\neg f$
- Conjunction: $f \wedge g$
- Disjunction: $f \vee g$
- Implication: $f \rightarrow g$
- Biconditional: $f \leftrightarrow g$

- The building blocks of the syntax are the propositional symbols and connectives. The set of propositional symbols can be anything (e.g., A, Wet, etc.), but the set of connectives is fixed to these five.
- All the propositional symbols are **atomic formulas** (also called atoms). We can **recursively** create larger formulas by combining smaller formulas using connectives.

Syntax of propositional logic

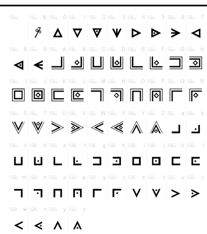
- Formula: A
- Formula: $\neg A$
- Formula: $\neg B \rightarrow C$
- Formula: $\neg A \wedge (\neg B \rightarrow C) \vee (\neg B \vee D)$
- Formula: $\neg\neg A$
- Non-formula: $A\neg B$
- Non-formula: $A + B$

- Here are some examples of valid and invalid propositional formulas.

Syntax of propositional logic

Key idea: syntax provides symbols

Formulas by themselves are just symbols (syntax). No meaning yet (semantics)!



- It's important to remember that whenever we talk about syntax, we're just talking about symbols; we're not actually talking about what they mean — that's the role of semantics. Of course it will be difficult to ignore the semantics for propositional logic completely because you already have a working knowledge of what the symbols mean.

Lecture

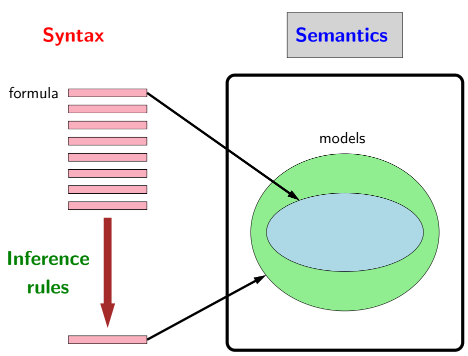
Overview

Propositional logic syntax

Propositional logic semantics

Inference rules

Propositional logic



- Having defined the syntax of propositional logic, let's talk about their semantics or meaning.

Model

Definition: model

A **model** w in propositional logic is an **assignment** of truth values to propositional symbols.

Example:

3 propositional symbols; A, B, C:

- $2^3 = 8$ possible models w:

$$\{A:0, B:0, C:0\}$$
$$\{A:0, B:0, C:1\}$$
$$\{A:0, B:1, C:0\}$$
$$\{A:0, B:1, C:1\}$$
$$\{A:1, B:0, C:0\}$$
$$\{A:1, B:0, C:1\}$$
$$\{A:1, B:1, C:0\}$$
$$\{A:1, B:1, C:1\}$$

- In logic, the word **model** has a special meaning, quite distinct from the way we've been using it in the class (quite an unfortunate collision). A model (in the logical sense) represents a possible state of affairs in the world. In propositional logic, this is an assignment that specifies a truth value (true or false) for each propositional symbol.

Interpretation function

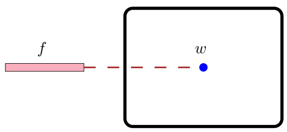
Definition: interpretation function

Let f be a formula.

Let w be a model.

An **interpretation function** $\mathcal{I}(f, w)$ returns:

- true (1) (say that w satisfies f)
- false (0) (say that w does not satisfy f)



- The semantics is given by an **interpretation function**, which takes a formula f and a model w, and returns whether w satisfies f. In other words, is f true in w?
- For example, if f represents "it is Wednesday" and w corresponds to right now, then $\mathcal{I}(f, w) = 1$. If w corresponded to yesterday, then $\mathcal{I}(f, w) = 0$.

Interpretation function: definition

Base case:

- For a propositional symbol p (e.g., A, B, C): $\mathcal{I}(p, w) = w(p)$

Recursive case:

- For any two formulas f and g, define:

$\mathcal{I}(f,w)$	$\mathcal{I}(g,w)$	$\mathcal{I}(\neg f,w)$	$\mathcal{I}(f \wedge g,w)$	$\mathcal{I}(f \vee g,w)$	$\mathcal{I}(f \rightarrow g,w)$	$\mathcal{I}(f \leftrightarrow g,w)$
0	0	1	0	0	1	1
0	1	1	0	1	1	0
1	0	0	0	1	0	0
1	1	0	1	1	1	1

- The interpretation function is defined recursively, where the cases neatly parallel the definition of the syntax.
- Formally, for propositional logic, the interpretation function is fully defined as follows. In the base case, the interpretation of a propositional symbol p is just gotten by looking p up in the model w. For every possible value of $(\mathcal{I}(f, w), \mathcal{I}(g, w))$, we specify the interpretation of the combination of f and g.

Interpretation function: example

Example: interpretation function

Formula: $f = (\neg A \wedge B) \leftrightarrow C$

Model: $w = \{A : 1, B : 1, C : 0\}$

Interpretation:

$$\mathcal{I}((\neg A \wedge B) \leftrightarrow C, w) = 1$$

$$\mathcal{I}(\neg A \wedge B, w) = 0 \qquad \mathcal{I}(C, w) = 0$$

$$\mathcal{I}(\neg A, w) = 0 \qquad \mathcal{I}(B, w) = 1$$

$$\mathcal{I}(A, w) = 1$$

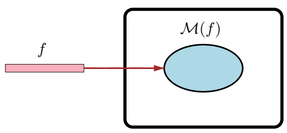
- For example, given the formula, we break down the formula into parts, recursively compute the truth value of the parts, and then finally combines these truth values based on the connective.

Formula represents a set of models

So far: each formula f and model w has an interpretation $\mathcal{I}(f, w) \in \{0, 1\}$

Definition: models

Let $\mathcal{M}(f)$ be the set of **models** w for which $\mathcal{I}(f, w) = 1$.



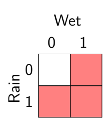
- So far, we've focused on relating a single model. A more useful but equivalent way to think about semantics is to think about the formula $\mathcal{M}(f)$ as **a set of models** — those for which $\mathcal{I}(f, w) = 1$.

Models: example

Formula:

$$f = \text{Rain} \vee \text{Wet}$$

Models:

$\mathcal{M}(f) =$ 

Key idea: compact representation

A **formula** *compactly* represents a set of **models**.

- In this example, there are three models for which the formula holds, as one can easily verify. From the point of view of $\mathcal{M}$, a formula's main job is to define a set of models.
- Recall that a model is a possible configuration of the world. So a formula like "it is raining" will pick out all the hypothetical configurations of the world where it's raining; in some of these configurations, it will be Wednesday; in others, it won't.

Knowledge base

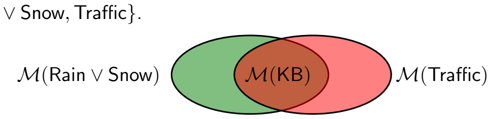
Definition: Knowledge base

A **knowledge base** KB is a set of formulas representing their conjunction / intersection:

$$\mathcal{M}(\text{KB}) = \bigcap_{f \in \text{KB}} \mathcal{M}(f).$$

Intuition: KB specifies constraints on the world. $\mathcal{M}(\text{KB})$ is the set of all worlds satisfying those constraints.

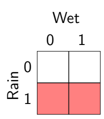
Let KB $= \{\text{Rain} \vee \text{Snow}, \text{Traffic}\}$.

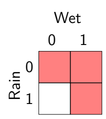


- If you take a set of formulas, you get a **knowledge base**. Each knowledge base defines a set of models — exactly those which are satisfiable by all the formulas in the knowledge base.
- Think of each formula as a fact that you know, and the **knowledge** is just the collection of those facts. Each fact narrows down the space of possible models, so the more facts you have, the fewer models you have.

Knowledge base: example

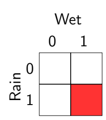
$\mathcal{M}(\text{Rain})$ $\quad\quad$ $\mathcal{M}(\text{Rain} \rightarrow \text{Wet})$





Intersection:

$\mathcal{M}(\{\text{Rain}, \text{Rain} \rightarrow \text{Wet}\})$



- As a concrete example, consider the two formulas Rain and Rain $\rightarrow$ Wet. If you know both of these facts, then the set of models is constrained to those where it is raining and wet.

Adding to the knowledge base

Adding more formulas to the knowledge base:

$$\text{KB} \longrightarrow \text{KB} \cup \{f\}$$

Shrinks the set of models:

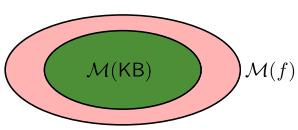
$$\mathcal{M}(\text{KB}) \longrightarrow \mathcal{M}(\text{KB}) \cap \mathcal{M}(f)$$

How much does $\mathcal{M}(\textbf{KB})$ shrink?

[whiteboard]

- We should think about a knowledge base as carving out a set of models. Over time, we will add additional formulas to the knowledge base, thereby winnowing down the set of models.
- Intuitively, adding a formula to the knowledge base imposes yet another constraint on our world, which naturally decreases the set of possible worlds.
- Thus, as the number of formulas in the knowledge base gets larger, the set of models gets smaller.
- A central question is how much f shrinks the set of models. There are three cases of importance.

Entailment



Intuition: f added no information/constraints (it was already known).

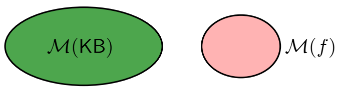
Definition: entailment

KB entails f (written KB $\models f$) iff $\mathcal{M}(\text{KB}) \subseteq \mathcal{M}(f)$.

Example: Rain $\wedge$ Snow $\models$ Snow

- The first case is if the set of models of f is a superset of the models of KB, then f adds no information. We say that KB **entails** f.

Contradiction



Intuition: f contradicts what we know (captured in KB).

Definition: contradiction

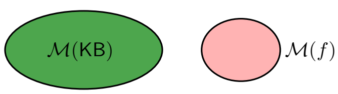
KB contradicts f iff $\mathcal{M}(\text{KB}) \cap \mathcal{M}(f) = \emptyset$.

Example: Rain $\wedge$ Snow contradicts $\neg$Snow

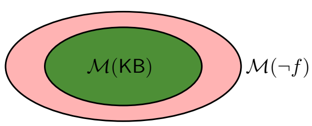
- The second case is if the set of models defined by f is completely disjoint from those of KB. Then we say that the KB and f **contradict** each other. If we believe KB, then we cannot possibly believe f.

Contradiction and entailment

Contradiction:



Entailment:

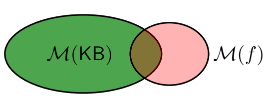


Proposition: contradiction and entailment

KB **contradicts** f iff KB **entails** $\neg f$.

- There is a useful connection between entailment and contradiction. If f is contradictory, then its negation ($\neg f$) is entailed, and vice-versa.
- You can see this because the models $\mathcal{M}(f)$ and $\mathcal{M}(\neg f)$ partition the space of models.

Contingency



Intuition: f adds non-trivial information to KB

$$\emptyset \subsetneq \mathcal{M}(\text{KB}) \cap \mathcal{M}(f) \subsetneq \mathcal{M}(\text{KB})$$

Example: Rain and Snow

- In the third case, we have a non-trivial overlap between the models of KB and f. We say in this case that f is **contingent**; f could be satisfied or not satisfied depending on the model.

Tell operation

Tell[f] → $\boxed{\text{KB}}$ → ?

Tell: *It is raining.*

$$\text{Tell}[\text{Rain}]$$

Possible responses:

- Already knew that: entailment (KB $\models f$)
- Don't believe that: contradiction (KB $\models \neg f$)
- Learned something new (update KB): contingent

- Having defined the three possible relationships that a new formula f can have with respect to a knowledge base KB, let's try to determine the appropriate response that a system should have.
- Suppose we tell the system that it is raining ($f = \text{Rain}$). If f is entailed, then we should reply that we already knew that. If f contradicts the knowledge base, then we should reply that we don't believe that. If f is contingent, then this is the interesting case, where we have non-trivially restricted the set of models, so we reply that we've learned something new.

Ask operation

Ask[f] → $\boxed{\text{KB}}$ → ?

Ask: *Is it raining?*

$$\text{Ask}[\text{Rain}]$$

Possible responses:

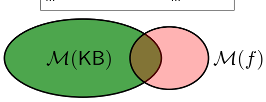
- Yes: entailment (KB $\models f$)
- No: contradiction (KB $\models \neg f$)
- I don't know: contingent

- Suppose now that we ask the system a question: is it raining? If f is entailed, then we should reply with a definitive yes. If f contradicts the knowledge base, then we should reply with a definitive no. If f is contingent, then we should just confess that we don't know.

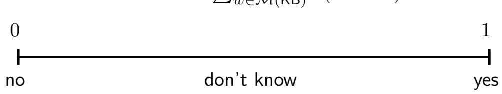
Digression: probabilistic generalization

Bayesian network: distribution over assignments (models)

w	$\mathbb{P}(W = w)$
{ A: 0, B: 0, C: 0 }	0.3
{ A: 0, B: 0, C: 1 }	0.1
...	...



$$\mathbb{P}(f \mid \text{KB}) = \frac{\sum_{w \in \mathcal{M}(\text{KB} \cup \{f\})} \mathbb{P}(W = w)}{\sum_{w \in \mathcal{M}(\text{KB})} \mathbb{P}(W = w)}$$



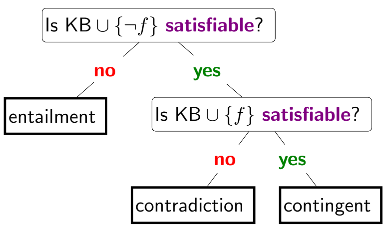
- Note that logic captures uncertainty in a very crude way. We can't say that we're almost sure or not very sure or not sure at all.
- Probability can help here. Remember that a Bayesian network (or more generally a factor graph) defines a distribution over assignments to the variables in the Bayesian network. Then we could ask questions such as: conditioned on having a cough but not itchy eyes, what's the probability of having a cold?
- Recall that in propositional logic, models are just assignments to propositional symbols. So we can think of KB as the evidence that we're conditioning on, and f as the query.

Satisfiability

Definition: satisfiability

A knowledge base KB is **satisfiable** if $\mathcal{M}(\text{KB}) \neq \emptyset$.

Reduce Ask$[f]$ and Tell$[f]$ to satisfiability:



- Now let's return to pure logic land again. How can we go about actually checking entailment, contradiction, and contingency? One useful concept to rule them all is **satisfiability**.
- Recall that we said a particular model w satisfies f if the interpretation function returns true $\mathcal{I}(f, w) = 1$. We can say that a formula f by itself is satisfiable if there is some model that satisfies f. Finally, a knowledge base (which is no more than just the conjunction of its formulas) is satisfiable if there is some model that satisfies all the formulas $f \in$ KB.
- With this definition in hand, we can implement Ask$[f]$ and Tell$[f]$ as follows:
- First, we check if KB $\cup \{\neg f\}$ is satisfiable. If the answer is no, that means the models of $\neg f$ and KB don't intersect (in other words, the two contradict each other). Recall that this is equivalent to saying that KB entails f.
- Otherwise, we need to do another test: check whether KB $\cup \{f\}$ is satisfiable. If the answer is no here, then KB and f are contradictory. Otherwise, we have that both f and $\neg f$ are compatible with KB, so the result is contingent.

Model checking

Checking satisfiability (SAT) in propositional logic is special case of solving CSPs!

Mapping:

propositional symbol	$\Rightarrow$	variable
formula	$\Rightarrow$	constraint
model	$\Leftarrow$	assignment

- Now we have reduced the problem of working with knowledge bases to checking satisfiability. The bad news is that this is an (actually, the canonical) NP-complete problem, so there are no efficient algorithms in general.
- The good news is that people try to solve the problem anyway, and we actually have pretty good SAT solvers these days. In terms of this class, this problem is just a CSP, if we convert the terminology: Each propositional symbol becomes a variable and each formula is a constraint. We can then solve the CSP, which produces an assignment, or in logic-speak, a model.

Model checking

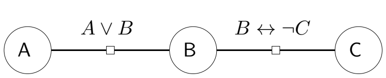
Example: model checking

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

Propositional symbols (CSP variables):

$$\{A, B, C\}$$

CSP:



Consistent assignment (satisfying model):

$$\{A : 1, B : 0, C : 1\}$$

- As an example, consider a knowledge base that has two formulas and three variables. Then the CSP is shown. Solving the CSP produces a consistent assignment (if one exists), which is a model that satisfies KB.
- Note that in the knowledge base tell/ask application, we don't technically need the satisfying assignment. An assignment would only offer a counterexample certifying that the answer **isn't** entailment or contradiction. This is an important point: entailment and contradiction is a claim about all models, not about the existence of a model.

Model checking

Definition: model checking

Input: knowledge base KB
Output: exists satisfying model ($\mathcal{M}(\text{KB}) \neq \emptyset$)?

Popular algorithms:

- DPLL (backtracking search + pruning)
- WalkSat (randomized local search)

Next: Can we exploit the fact that factors are formulas?

- Checking satisfiability of a knowledge base is called **model checking**. For propositional logic, there are several algorithms that work quite well which are based on the algorithms we saw for solving CSPs (backtracking search and local search).
- However, can we do a bit better? Our CSP factors are not arbitrary — they are logic formulas, and recall that formulas are defined recursively and have some compositional structure. Let's see how to exploit this.

Lecture

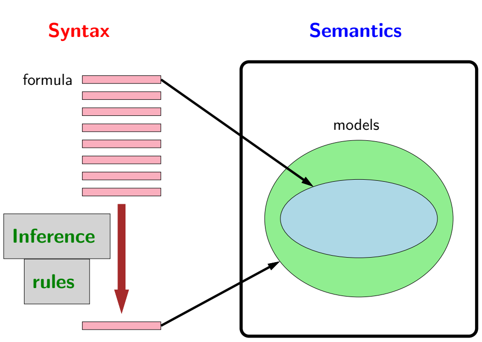
Overview

Propositional logic syntax

Propositional logic semantics

Inference rules

Propositional logic



- 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.

Inference rules

Example of making an inference:

It is raining. (Rain)
If it is raining, then it is wet. (Rain $\rightarrow$ Wet)
Therefore, it is wet. (Wet)

$$\frac{\text{Rain}, \quad \text{Rain} \rightarrow \text{Wet}}{\text{Wet}} \qquad \frac{\text{(premises)}}{\text{(conclusion)}}$$

Definition: Modus ponens inference rule

For any propositional symbols p and q:

$$\frac{p, \quad p \rightarrow q}{q}$$

- The idea of making an inference should be quite intuitive to you. The classic example is **modus ponens**, which captures the if-then reasoning pattern.

Inference framework

Definition: inference rule

If $f_1, \dots, f_k, g$ are formulas, then the following is an **inference rule**:

$$\frac{f_1, \quad \dots \quad, f_k}{g}$$

Key idea: inference rules

Rules operate directly on **syntax**, not on **semantics**.

- In general, an inference rule has a set of premises and a conclusion. The rule says that if the premises are in the KB, then you can add the conclusion to the KB.
- We haven't yet specified whether this is a valid thing to do, but it is a thing to do. Remember, syntax is just about symbol pushing; it is only by linking to models that we have notions of truth and meaning (semantics).

Inference algorithm

Algorithm: forward inference

Input: set of inference rules Rules.
Repeat until no changes to KB:
    Choose set of formulas $f_1, \dots, f_k \in$ KB.
    If matching rule $\frac{f_1, \quad \dots \quad, f_k}{g}$ exists:
        Add g to KB.

Definition: derivation

KB **derives/proves** f (KB $\vdash f$) iff f eventually gets added to KB.

- Given a set of inference rules (e.g., modus ponens), we can just keep on trying to apply rules. Those rules generate new formulas which get added to the knowledge base, and those formulas might then be premises of other rules, which in turn generate more formulas, etc.
- We say that the KB derives or proves a formula f if by blindly applying rules, we can eventually add f to the KB.

Inference example

Example: Modus ponens inference

Starting point:

KB $=$ {Rain, Rain $\rightarrow$ Wet, Wet $\rightarrow$ Slippery}

Apply modus ponens to Rain and Rain $\rightarrow$ Wet:

KB $=$ {Rain, Rain $\rightarrow$ Wet, Wet $\rightarrow$ Slippery, Wet}

Apply modus ponens to Wet and Wet $\rightarrow$ Slippery:

KB $=$ {Rain, Rain $\rightarrow$ Wet, Wet $\rightarrow$ Slippery, Wet, Slippery}

Converged.

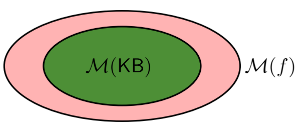
Can't derive some formulas: $\neg$Wet, Rain $\rightarrow$ Slippery

- Here is an example where we've applied modus ponens twice. Note that Wet and Slippery are derived by the KB.
- But there are some formulas which cannot be derived. Some of these underivable formulas will look bad anyway ($\neg$Wet), but others will seem reasonable (Rain $\rightarrow$ Slippery).

Desiderata for inference rules

Semantics

Interpretation defines **entailed/true** formulas: KB $\models f$:



Syntax:

Inference rules **derive** formulas: KB $\vdash f$

How does $\{f :$ KB $\models f\}$ relate to $\{f :$ KB $\vdash f\}$?

- We can apply inference rules all day long, but now we desperately need some guidance on whether a set of inference rules is doing anything remotely sensible.
- For this, we turn to semantics, which gives an objective notion of truth. Recall that the semantics provides us with $\mathcal{M}$, the set of satisfiable models for each formula f or knowledge base. This defines a set of formulas $\{f :$ KB $\models f\}$ which are defined to be true.
- On the other hand, inference rules also gives us a mechanism for generating a set of formulas, just by repeated application. This defines another set of formulas $\{f :$ KB $\vdash f\}$.

Truth

$\{f :$ KB $\models f\}$

- Imagine a glass that represents the set of possible formulas entailed by the KB (these are necessarily true).
- By applying inference rules, we are filling up the glass with water.

Soundness

Definition: soundness

A set of inference rules Rules is sound if:

$$\{f : \text{KB} \vdash f\} \subseteq \{f : \text{KB} \models f\}$$

- We say that a set of inference rules is **sound** if using those inference rules, we never overflow the glass: the set of derived formulas is a subset of the set of true/entailed formulas.

Completeness

Definition: completeness

A set of inference rules Rules is complete if:

$$\{f : \text{KB} \vdash f\} \supseteq \{f : \text{KB} \models f\}$$

- We say that a set of inference rules is **complete** if using those inference rules, we fill up the glass to the brim (and possibly go over): the set of derived formulas is a superset of the set of true/entailed formulas.

Soundness and completeness

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

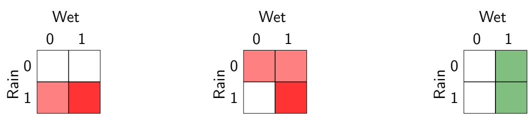
- **Soundness**: nothing but the truth
- **Completeness**: whole truth

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

Soundness: example

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

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



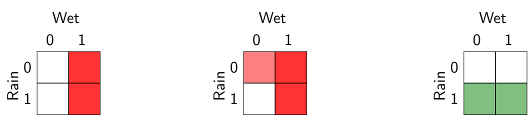
Sound!

- 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 $\to$ 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 $\frac{\text{Wet}, \quad \text{Rain} \to \text{Wet}}{\text{Rain}}$ sound?

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



Unsound!

- Here is another example: given Wet and Rain $\to$ 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:

$\text{KB} = \{\text{Rain}, \text{Rain} \lor \text{Snow} \to \text{Wet}\}$

$f = \text{Wet}$

$\text{Rules} = \{\frac{f, \quad f \to g}{g}\}$ (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

$\Downarrow$

propositional logic with only Horn clauses

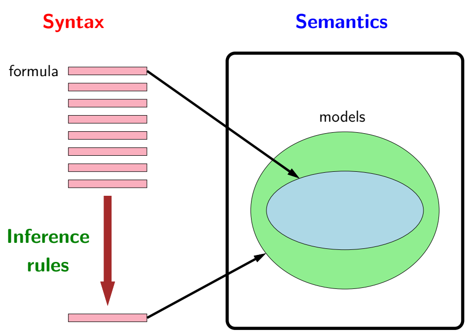
Option 2: Use more powerful inference rules

Modus ponens

$\Downarrow$

resolution

Summary



- 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.