CSC 665: Artificial Intelligence

Lecture: Logic II

1 Concepts for inference

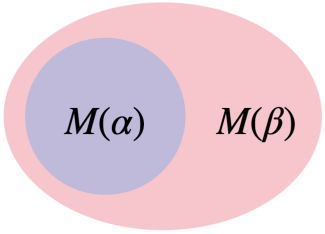
Last time we took an extensive look at representation for logical approaches to artificial intelligence. Most of the work went into describing the syntax and semantics of a simple but powerful logical language: propositional logic. Using propositional logic as an expressive tool, we were then able to encode known facts about the real world as logical formulas in a knowledge base. In this lecture we begin our study of inference algorithms for answering useful questions given such a knowledge base.

We begin by defining the key concepts that we'll need to frame inference; namely, entailment and contradiction.

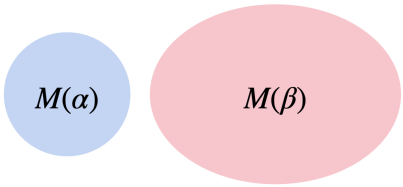
Definition: A formula α entails a formula β if every model that satisfies α also satisfies β. Notationally, we write

$$\alpha \models \beta \text{ iff } M(\alpha) \subseteq M(\beta).$$

Intuitively, α entails β if β is true in every world in which α is true.



Definition: A formula α contradicts a formula β if there is no model that satisfies both α and β, i.e. $M(\alpha) \cap M(\beta) = \emptyset$. Intuitively, α and β contradict each other if there is no world in which both α and β are both simultaneously true.



The inference task we are interested in is to determine whether a formula α is entailed or contradicted by our knowledge base KB.

Note that it is possible for a formula to be neither entailed by nor in contradiction with a knowledge base. (This third case is sometimes called contingency.)

2 Model checking

Our first inference algorithm directly operationalizes the definition of entailment. The definition says that we need to check whether one set of models is contained in another. So our first inference procedure simply iterates through all possible models and checks whether this is the case. The name of this algorithm is, appopriately, model checking. The pseudocode for checking whether a formula α is entailed by a knowledge base KB is given in Algorithm 1.

Algorithm 1 Model checking

```
1: procedure CheckModel(KB, α)
2:     for every possible model m do
3:         if m satisfies KB then
4:             if m does not satisfy α then
5:                 return False
6:     return True
```

How should we evaluate this algorithm? Recall from our discussion of search algorithms that we can break this question down into an assessment of the algorithm's correctness and an assessment of its efficiency.

First, efficiency. The algorithm loops through all possible models. From last time, we know that a knowledge base with n propositional symbols has 2^n models. Thus the time complexity of model checking is $O(2^n)$. From a space perspective, we only need to store a representation of the current model m that we are checking. A model is simply an assignment of truth values to all n symbols, so the space complexity is $O(n)$.

Our framework for evaluating the correctness of logical inference procedures requires two new definitions.

Definition: An inference procedure is sound if it only derives formulas that are entailed by the knowledge base. That is, the set of formulas derived by the procedure is a subset of all formulas that are actually entailed by KB.

Definition: An inference procedure is complete if it derives all formulas entailed by the knowledge base (and possibly formulas that are not entailed by the knowledge base). That is, the set of formulas that are actually entailed by KB is a subset of the formulas derived by the procedure.

Ideally we want a procedure that is both sound and complete. Such a procedure would give us the truth, the whole truth (completeness), and nothing but the truth (soundness). Note that achieving just soundness or just completeness without the other is actually quite easy, as the next examples show.

Example: The constant function $f(\text{KB}, \alpha) = \text{False}$ is sound but not complete. According to f, no formula is entailed by KB, so it derives the empty set. Since the empty set is a subset of every set, in particular it is a subset of the formulas entailed by KB, thus satisfying the definition of soundness.

Example: The constant function $g(\text{KB}, \alpha) = \text{True}$ is complete but not sound. According to g, every formula is entailed by KB, so it derives the universal set of all possible formulas. Since every set is a subset of the universal set, the set of formulas entailed by KB are a particular subset, thus satisfying the definition of completeness.

Model checking is both sound (because of the checks on lines 3 and 4) and complete (because of the exhaustive loop on line 2), so it is an ideal algorithm with respect to correctness. The exponential time complexity, however, is motivation to find alternative inference procedures that are hopefully faster.

3 Inference rules

3.1 Modus ponens on propositional logic

Model checking operates in the world of semantics, looping through models and checking whether they satisfy certain formulas of interest. An alternative approach is to work at a purely syntactic level, performing abstract symbol manipulations to derive new formulas from old ones. This motivates the development of inference rules.

Definition: One of the oldest and most natural inference rules is known as modus ponens:

$$\frac{p,\ p \implies q}{q},$$

where p and q are atomic symbols. This rule says that given formulas p and $p \implies q$, we derive the formula q. In general, all inference rules have the form

$$\frac{f_1, \ldots, f_n}{g},$$

for formulas $f_1, \ldots, f_n, g$.

If we have a set of inference rules and a knowledge base, we can derive new formulas by repeatedly applying the inference rules to matching formulas in the knowledge base, adding the new formulas to the knowledge base as we go. This is the idea behind forward inference with a rule set R (Algorithm 2).

Algorithm 2 Forward inference

```
1: procedure ForwardInference(KB, R)
2:     while we can keep adding new formulas to KB do
3:         choose formulas f_1, ..., f_n ∈ KB
4:         if there is a matching inference rule (f_1, ..., f_n)/g ∈ R then
5:             add g to KB
6:     return KB
```

Example: Suppose we run forward inference with a rule set consisting of just modus ponens. If our initial knowledge base is

$$\text{KB} = \{R,\ R \implies W,\ W \implies S\},$$

then one application of modus ponens on the first two formulas allows us to derive the formula W, yielding the augmented knowledge base

$$\text{KB} = \{R,\ R \implies W,\ W \implies S,\ W\}.$$

A further application of modus ponens allows us to derive S, yielding

$$\text{KB} = \{R,\ R \implies W,\ W \implies S,\ W,\ S\}.$$

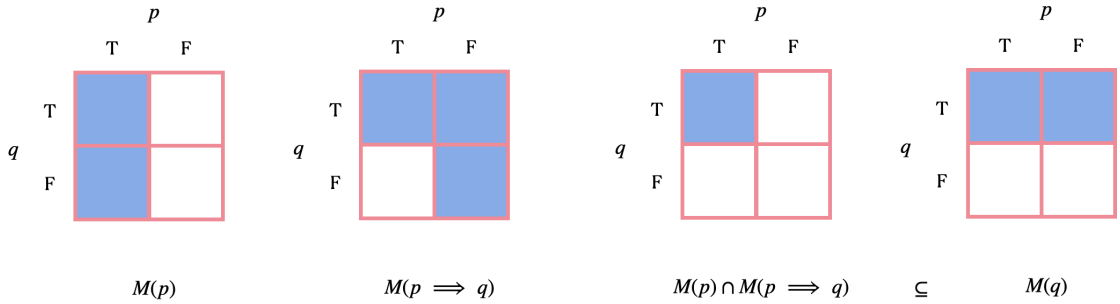
At this point, modus ponens can take us no further and forward inference terminates. [1]

A key advantage of forward inference is that by working purely at the level of syntax, we can ignore irrelevant information when deriving new formulas. For example, suppose there were 100 other additional symbols in the knowledge base above. Forward inference would still have no trouble deriving W and S, assuming that it can successfully find the relevant formulas to match the two antecedents of the modus

[1] Since forward inference requires no knowledge of semantics or real world interpretation, we can treat the expressions above as abstract strings of pure symbols. But if you'd like to ground this example in the real world, you can imagine that R stand for rainy, W stands for wet, and S stands for slippery.

ponens rule. Meanwhile model checking would have to explicitly loop through all 2^{100} models in order to check entailment, which is computationally intractable.

We now consider the correctness of forward inference. To determine whether forward inference is sound, it suffices to check the soundness of each inference rule alone. Fortunately, modus ponens is indeed a sound inference rule, as the following picture proof demonstrates.



Example: Not all inference rules are sound. Draw a similar picture proof as above to convince yourself that the following inference rule is not sound.

$$\frac{q,\ p \implies q}{p}.$$

The soundness of modus ponens guarantees the soundness of forward inference with modus ponens. Unfortunately, forward inference with modus ponens is not complete. To see why, consider the knowledge base $\text{KB} = \{R,\ (R \vee S) \implies W\}$ and the formula α. Then although $\text{KB} \models \alpha$, forward inference is unable to derive α using modus ponens alone. This highlights the limitations of treating formulas as pure syntax.

Given that forward inference with modus ponens is not complete, how should we proceed if we want completeness in our inference procedures? There are two paths we can take:

1. **Simplify our representation.** If we restrict ourselves to a less expressive language than propositional logic, then modus ponens will be powerful enough to yield a complete inference algorithm on this smaller language.
2. **Use more powerful inference rules.** If modus ponens alone is not enough for propositional logic, then we will need to consider additional inference rules to obtain a complete inference procedure.

3.2 Modus ponens on Horn clauses

Here we describe the first approach of simplifying our language for representation. Next lecture we will see what missing inference rules we can add if we don't wish to restrict ourselves to a less expressive representational form. We begin with a slurry of definitions.

Definition: A literal is either an atomic symbol or its negation.

Definition: A clause is a disjunction of literals.

Definition: A definite clause is a formula of the form

$$(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \implies q,$$

where $p_1, \ldots, p_n, q$ are all literals. Equivalently, a definite clause can be written as

$$\neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n \vee q,$$

which shows that a definite clause is a clause with exactly one literal that is not a negation.

Definition: A goal clause is a formula of the form

$$(p_1 \wedge p_2 \wedge \cdots \wedge p_n) \implies \text{False},$$

where $p_1, \ldots, p_n, q$ are all literals. Equivalently, a goal clause can be written as

$$\neg p_1 \vee \neg p_2 \vee \cdots \vee \neg p_n,$$

which shows that a goal clause is a clause where all the literals are negations.

Definition: A Horn clause is either a definite clause or a goal clause. Equivalently, a Horn clause is a clause with at most one literal that is not a negation.

Example: Below we indicate whether the given formulas are literals, clauses, definite clauses, goal clauses, or Horn clauses. Be sure that you understand each of these examples.

formula	**literal**?	**clause**?	**definite clause?**	**goal clause?**	**Horn clause?**
A	✓	✓	✓		✓
$\neg A$	✓	✓		✓	✓
$A \vee B$		✓			
$\neg A \vee B$		✓	✓		✓
$\neg A \vee \neg B$		✓		✓	✓
$A \wedge B$					

The final definition above for Horn clauses is the key for restricting our language. We will consider the subset of propositional logic in which we only allow formulas that are Horn clauses. In addition to working with this simpler language, we will also add the following generalization of modus ponens.

Definition: Generalizing modus ponens to allow for implications with multiple premises gives the following inference rule:

$$\frac{p_1, \ldots, p_n, (p_1 \wedge \cdots \wedge p_n) \implies q}{q}.$$

Theorem: Forward inference with generalized modus ponens is sound and complete on a knowledge base of Horn clauses.

Thus restricting ourselves to the smaller language of Horn clauses allowed us to make a tradeoff between expressivity and completeness. Note that there are formulas in propositional logic that cannot be expressed using Horn clauses alone, so we are giving up some expressive power. But in return we get the guarantee that forward inference is complete.

Furthermore, inference can be done in linear time with respect to the number of atomic symbols. So in cases where we can get a way with working exclusively with Horn clauses, modus ponens is as good as we could have hoped for.

But in many cases we will not have a knowledge base consisting of Horn clauses alone. Next time we will see what additional inference rules are required to get an algorithm that is complete on all of propositional logic.