CSC 665: Artificial Intelligence

Lecture: Logic I

1 Why logic?

Search is a powerful paradigm for designing agents capable of rational behavior, but not everything is a search problem.

Consider the following dialog with a chatbot.

Hello! Talk to me in English. Tell me something new (end the sentence with '.') or ask me a question (end the sentence with '?'). Type 'help' for additional commands.

Garfield is a cat?

I don't know.

Garfield is a cat.

I learned something.

Garfield is a cat?

Yes.

Garfield is a mammal?

I don't know.

Every cat is a mammal.

I learned something.

Garfield is a mammal?

Yes.

Jon feeds Garfield.

I learned something.

Jon feeds some cat?

Yes.

Jon likes Garfield?

I don't know.

If a person feeds a cat then the former likes the latter.

I learned something.

Jon likes Garfield?

I don't know.

Jon is a person.

I learned something.

Jon likes Garfield?

Yes.

The agent is capable of making some impressive logical inferences, putting together multiple different facts to derive correct conclusions. The agent is also aware of the limits of its knowledge, and it tells us when it doesn't know something. It's certainly not clear how a search-based agent could achieve this level of logical reasoning. It seems that we need something different.

2 Representation

As always, we begin with representation (modeling). Different logical languages have different levels of expressivity. To specify a language, we need to specify two things:

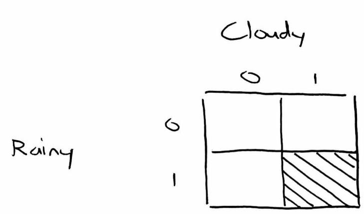
1. **Syntax:** defines the set of valid formulas in the language. *What expressions are legal?*
2. **Semantics:** for each formula, specify the set of models (configurations of the world). *What do these expressions mean?*

Example: In grade-school algebra, $x + y = 4$ is a formula with valid syntax, whereas $4 = +$ is not.

Example: The logical formula

$$\text{Rainy} \wedge \text{Cloudy}$$

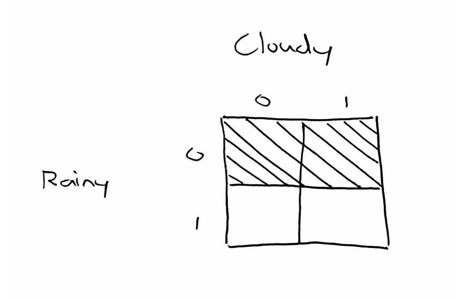
is syntactically correct and corresponds semantically to the following configuration of the world



Similarly, the logical formula

$$\neg\text{Rainy}$$

is syntactically correct and corresponds semantically to the following two configurations of the world



Example: Here are two formulas with different syntax but identical semantics:

$$x = 2 + 3$$
$$x = 3 + 2$$

Within a single language, a given expression must have unambiguous semantics. But the same formula might have different semantics in different languages. For example the expression

```
3 / 2
```

means one thing (integer division) in Python 2 and another (floating point division) in Python 3.

The first logical language we will study in this unit is called *propositional logic*. First we describe its syntax, then we move on to its semantics.

2.1 Propositional logic: syntax

A formula in propostional logic consists of symbols and connectives.

1. **Symbols:** In addition to atomic symbols such as $A, B, C, \text{Cloudy}, \text{Rainy}$, and so on, we have two reserved symbols: True and False.
2. **Connectives:**
   - $\neg$ (not, negation)
   - $\wedge$ (and, conjunction)
   - $\vee$ (or, disjunction)
   - $\implies$ (implies, implication)
   - $\iff$ (if and only if, biconditional equivalence)

From your experience with programming languages, you are probably already familiar with the rules for stringing together symbols and connectives to form valid logical formulas. The following examples and non-examples should not be too surprising.

Example: The following formulas are all valid according to the syntax of propositional logic.

- A
- $\neg A$
- $\neg B \implies C$
- $\neg\neg B$
- $(\neg A \wedge (\neg B \implies C)) \iff (C \vee D)$

The following sequences of symbols and connectives are *not* valid formulas.

- $A \neg B$
- AB
- $A + B$
- $\implies B$

We can be more precise and complete about the syntax of propositional logic by writing down a grammar in Backus-Naur form (BNF). You may be familiar with BNF if you've taken a course in programming languages.

$$
\begin{array}{rcl}
\textit{Sentence} & \rightarrow & \textit{AtomicSentence} \mid \textit{ComplexSentence} \\
\textit{AtomicSentence} & \rightarrow & \textit{True} \mid \textit{False} \mid P \mid Q \mid R \mid \ldots \\
\textit{ComplexSentence} & \rightarrow & \textbf{(} \ \textit{Sentence} \ \textbf{)} \\
& \mid & \neg \ \textit{Sentence} \\
& \mid & \textit{Sentence} \land \textit{Sentence} \\
& \mid & \textit{Sentence} \lor \textit{Sentence} \\
& \mid & \textit{Sentence} \ \Rightarrow \ \textit{Sentence} \\
& \mid & \textit{Sentence} \ \Leftrightarrow \ \textit{Sentence}
\end{array}
$$

2.2 Propositional logic: semantics

Definition: A model m is an assignment of truth values (True/False) to symbols.

Example: If we have three symbols A, B, C, then there are $2^3 = 8$ possible models. (Two possible truth values for each of three symbols.) Convince yourself by writing out all the possible models.

To link syntax and semantics, we need to say what the connectives mean. That is, we need to say what the truth values are for the most basic formula involving a single logical connective in all possible models. The easiest way to do this is via a *truth table*.

p	q	$\neg p$	$p \land q$	$p \lor q$	$p \implies q$	$p \iff q$
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

Now given a model m and any formula α, we can evaluate the truth of α in m.

Example: Suppose the model m is $\{A : \mathrm{T}, B : \mathrm{T}, C : \mathrm{F}\}$. Then the formula α given by

$$(\neg A \land B) \iff C$$

is true, as you should verify. We say that "m satisfies α" to mean that α is true in m.

Definition: $M(\alpha)$ is the set of all models that satisfy a formula α. That is, $M(\alpha) = \{m \mid m \text{ satisfies } \alpha\}$.

2.3 Knowledge bases

Now that we've described a simple logical language, i.e. propositional logic, we can create a representation of the real world by encoding known facts as formulas in the language. We call the collection of all such encoded facts a *knowledge base*.

Definition: A knowledge base is a conjunction of a set of formulas.

$$\mathrm{KB} = \alpha_1 \land \alpha_2 \land \alpha_3 \land \cdots \land \alpha_n$$

3 Inference

Once we've specified a language and used that language to encode facts about the world in a knowledge base, we can start to think about algorithms for making logical inferences from that knowledge base. There are two key types of inferences that we will be interested in:

1. Entailment
2. Contradiction

Example: Returning the language of grade-school algebra, suppose our knowledge base consists of the sole formula $x = 0$.

- Then the formula $xy = 0$ is *entailed* by our knowledge base, since a product of two numbers is always zero if one of the factors in the product zero.
- In contrast, the formula $xy = 3$ *contradicts* our knowledge base — there are no values of x and y that make both this formula and our knowledge base simultaneously true.
- It is possible to have a formula that neither is entailed by nor contradicts our knowledge base. For example, $x + y = 3$ is one such formula.

Next time, we will formalize the notions of entailment and contradiction, and begin to look at algorithms for making these types of inferences.