CS 188 Introduction to Artificial Intelligence
Spring 2023 — Note 6

These lecture notes are heavily based on notes originally written by Henry Zhu. All figures are from *Artificial Intelligence: A Modern Approach.*

Last updated: January 15, 2023

First-Order Logic

The second dialect of logic, **first-order logic (FOL)**, is more expressive than propositional logic and uses objects as its basic components. With first-order logic we can describe relationships between objects and apply functions to them. Each object is represented by a **constant symbol**, each relationship by a **predicate symbol**, and each function by a **function symbol**.

The following table summarizes the first order logic syntax.

Sentence	$\rightarrow$	*AtomicSentence* | *ComplexSentence*
AtomicSentence	$\rightarrow$	*Predicate* | *Predicate*(*Term*, . . .) | *Term* = *Term*
ComplexSentence	$\rightarrow$	(*Sentence*) | [*Sentence*]
	|	$\neg$ *Sentence*
	|	*Sentence* $\wedge$ *Sentence*
	|	*Sentence* $\vee$ *Sentence*
	|	*Sentence* $\Rightarrow$ *Sentence*
	|	*Sentence* $\Leftrightarrow$ *Sentence*
	|	*Quantifier Variable*, . . . *Sentence*
Term	$\rightarrow$	*Function*(*Term*, . . .)
	|	*Constant*
	|	*Variable*
Quantifier	$\rightarrow$	$\forall$ | $\exists$
Constant	$\rightarrow$	A | X_1 | *John* | $\cdots$
Variable	$\rightarrow$	a | x | s | $\cdots$
Predicate	$\rightarrow$	*True* | *False* | *After* | *Loves* | *Raining* | $\cdots$
Function	$\rightarrow$	*Mother* | *LeftLeg* | $\cdots$
OPERATOR PRECEDENCE	:	$\neg, =, \wedge, \vee, \Rightarrow, \Leftrightarrow$

Figure 8.3 The syntax of first-order logic with equality, specified in Backus–Naur form (see page 1060 if you are not familiar with this notation). Operator precedences are specified, from highest to lowest. The precedence of quantifiers is such that a quantifier holds over everything to the right of it.

Terms in first-order logic are logical expressions that refer to an object. The simplest form of terms are constant symbols. However we don't want to define distinct constant symbols for every possible object. For example, if we want to refer to John's left leg and Richard's left leg, we can do so by using **function symbols** like $Leftleg(John)$ and $Leftleg(Richard)$. Function symbols are just another way to name objects and are not actual functions.

Atomic sentences in first-order logic are descriptions of relationships between objects, and are true if the relationship holds. An example of an atomic sentence is $Brother(John, Richard)$ which is formed by a predicate symbol followed by a list of terms inside the parentheses. **Complex sentences** of first order logic are analogous to those in propositional logic and are atomic sentences connected by logical connectives.

Naturally we would like ways to describe entire collections of objects. For this we use **quantifiers**. The **universal quantifier** $\forall$, has the meaning "for all," and the **existential quantifier** $\exists$, has the meaning "there exists."

For example, if the set of objects in our world is the set of all debates, the sentence $\forall a,\ TwoSides(a)$ could be translated as "there are two sides to every debate". If the set of objects in our world is people, the sentence $\forall x, \exists y,\ SoulMate(x,y)$ would mean "for all people, there is someone out there who is their soulmate. The **anonymous variables** a,x,y are stand-ins for objects, and can be **substituted** for actual objects, for example, substituting Laura for x into our second example would result in a statement that "there is someone out there for Laura."

The universal and existential quantifiers are convenient ways to express a conjunction or disjunction, respectively, over all objects. It follows that they also obey De Morgan's laws (note the analogous relationship between conjunctions and disjunctions):

$$\begin{array}{lll}
\forall x\ \neg P & \equiv & \neg\exists x\ P \\
\neg\forall x\ P & \equiv & \exists x\ \neg P \\
\forall x\ P & \equiv & \neg\exists x\ \neg P \\
\exists x\ P & \equiv & \neg\forall x\ \neg P
\end{array}
\qquad\qquad
\begin{array}{lll}
\neg(P \vee Q) & \equiv & \neg P \wedge \neg Q \\
\neg(P \wedge Q) & \equiv & \neg P \vee \neg Q \\
P \wedge Q & \equiv & \neg(\neg P \vee \neg Q) \\
P \vee Q & \equiv & \neg(\neg P \wedge \neg Q)
\end{array}$$

Finally, we use the **equality symbol** to signify that two symbols refer to the same object. For example, the incredible sentence $(Wife(Einstein) = FirstCousin(Einstein) \wedge Wife(Einstein) = SecondCousin(Einstein))$ is true!

Unlike with propositional logic, where a model was an assignment of true or false to all proposition symbols, a model in first-order logic is a mapping of all constant symbols to objects, predicate symbols to relations between objects, and function symbols to functions of objects. A sentence is true under a model if the relations described by the sentence are true under the mapping. While the number of models of a propositional logical system is always finite, there may be an infinite number of models of a first order logical system if the number of objects is unconstrained.

These two dialects of logic allow us to describe and think about the world in different ways. With propositional logic, we model our world as a set of symbols that are true or false. Under this assumption, we can represent a possible world as a vector, with a 1 or 0 for every symbol. This binary view of the world is what is known as a **factored representation**. With first-order logic, our world consists of objects that relate to one another. This second object-oriented view of the world is known as a **structured representation**, is in many ways more expressive and is more closely aligned with the language we naturally use to speak about the world.

First Order Logical Inference

With first order logic we formulate inference exactly the same way. We'd like to find out if $KB \models q$, that is if q is true in all models under which KB is true. One approach to finding a solution is **propositionalization** or translating the problem into propositional logic so that it can be solved with techniques we have already laid out. Each universal (existential) quantifier sentence can be converted to a conjunction (disjunction) of sentences with a clause for each possible object that could be substituted in for the variable. Then, we can use a SAT solver, like DPLL or Walk-SAT, (un)satisfiability of $(KB \land \neg q)$.

One problem with this approach is there are an infinite number of substitutions that we could make, since there is no limit to how many times we can apply a function to a symbol. For example, we can nest the function $Classmate(\cdots Classmate(Classmate(Austen))\cdots)$ as many times as we'd like, until we reference the whole school. Luckily, a theorem proved by Jacques Herbrand (1930) tells us that if a sentence is entailed by a knowledge base that there is a proof involving just a *finite* subset of the propositionalized knowledge base. Therefore, we can try iterating through finite subsets, specifically searching via iterative deepening through nested function applications, i.e. first search through substitutions with constant symbols, then substitutions with $Classmate(Austen)$, then substitutions with $Classmate(Classmate(Austen))$, ...

Another approach is to directly do inference with first-order logic, also known as **lifted inference**. For example, we are given $(\forall x\, HasAbsolutePower(x) \land Person(x) \Rightarrow Corrupt(x)) \land Person(John) \land HasAbsolutePower(John)$ ("absolute power corrupts absolutely"). We can infer $Corrupt(John)$ by substituting x for $John$. This rule is known as **Generalized Modus Ponens**. The forward chaining algorithm for first order logic repeatedly applies generalized Modus Ponens and substitution to infer q or show that it cannot be inferred.

Logical Agents

Now that we understand how to formulate what we know and how to reason with it, we will talk about how to incorporate the power of deduction into our agents. One obvious ability an agent should have is the ability to figure out what state it is in, based on a history of observations and what it knows about the world (**state-estimation**). For example, if we told the agent that the air starts to shimmer near pools of lava and it observed that the air right before it is shimmering, it could infer that danger is nearby.

To incorporate its past observations into an estimate of where it currently is, an agent will need to have a notion of time and transitions between states. We call state attributes that vary with time **fluents** and write a fluent with an index for time, e.g. $Hot^t =$ the air is hot at time t. The air should be hot at time t if something causes the air to be hot at that time, or the air was hot at the previous time and no action occurred to change it. To represent this fact we can use the general form of the **successor-state axiom** below:

$$F^{t+1} \Leftrightarrow ActionCausesF^t \lor (F^t \land \neg ActionCausesNotF^t)$$

In our world, the transition could be formulated as $Hot^{t+1} \Leftrightarrow StepCloseToLava^t \lor (Hot^t \land \neg StepAwayFromLava^t)$.

Having written out the rules of the world in logic, we can now actually do planning by checking the satisfiability of some logic proposition! To do this, we construct a sentence that includes information about the initial state, the transitions (successor-state axioms), and the goal. (e.g. $InOasis^T \land Alive^T$ encodes the objective of surviving and ending up in the oasis by time T). If the rules of the world have been properly formulated, then finding a satisfying assignment to all the variables will allow us to extract a sequence of actions that will carry the agent to the goal.

Summary

In this note, we introduced the concept of logic which knowledge-based agents can use to reason about the world and make decisions. We introduced the language of logic, its syntax and the standard logical equivalences. Propositional logic is a simple language that is based on proposition symbols and logical connectives. First-order logic is a representation language more powerful than propositional logic. The syntax of first-order logic builds on that of propositional logic, using terms to represent objects and universal and existential quantifiers to make assertions.

We further described the DPLL algorithm used to check satisfiability (SAT problem) in propositional logic. It is a depth-first enumeration of possible models, using early termination, pure symbol heuristic and unit clause heuristic to improve performance. The forward chaining algorithm can be used for reasoning when our knowledge base consists solely of literals and implications in propositional logic.

Inference in first-order logic can be done directly by using rules like Generalized Modus Ponens or by propositionalization, which translates the problem into propositional logic and uses a SAT solver to draw conclusions.