CS 151, Spring 00
Logic 1

Margaret M. Fleck

February 2, 2000

Reading: Russell and Norvig, 6.1-6.3 and 7.1-7.2.

1 Why logic?

Logic is a system for encoding our knowledge and beliefs about the world in a form that the computer can manipulate. In order for the computer to handle the representations, they must be clearly defined and relatively simple.

It's not feasible to explicitly represent every fact about the world, or even every interesting fact. The world is far too big! Therefore, logical representations represent only some specific facts individually. To be able to deduce all the other facts, the logical system also includes (a) generalizations about how the world behaves and (b) precise rules for deriving facts from other facts (inference rules).

Logical representations do not necessarily describe the world accurately. Rather, in AI, we hope to make them match our human models of how the world works. Therefore, they may contain some generalizations that humans belief but which aren't completely accurate.

First-Order Predicate Calculus (FOPC) is a popular system for representing facts and generalizations, and rules for reasoning about them. It has a long history of success in mathematics. The basic machinery of FOPC is suitable for handling simple sets of facts and basic concepts. It becomes cumbersome if complex situations or sophisticated concepts are represented directly using the basic machinery.

To extend FOPC to the full range of (say) mathematics, one typically uses FOPC to define high-level terminology, additional notation, and inference rules that operate directly on these new objects. For example, we define limit and derivative notation to explain calculus. This augmented version of FOPC is then used to encode and reason about the sophisticated problems.

In FOPC, the truth of a sentence depends on whether the corresponding facts are true in the real world (or, for more abstract problems, true in some abstract model world). However, it is not at all clear how to tell a computer, in precise terms, how to connect high-level sentences such as "played (David, Warcraft)" to specific features of the real world.

In current AI research (also in areas such as the physical sciences), this connection is typically made using human intelligence. We know how to inspect the world and determine whether a specific type of action (e.g. walking) is happening. This allows AI researchers to develop reasoning algorithms without first solving the problems of sensory perception and robotics. However, this is clearly insufficient for an autonomous robot.

The alternative is to define a relationship between logic and the world for very basic concepts, then derive relationships for complex concepts. This is one of the traditional occupations of researchers in mathematical foundations, logic, and philosophy. It is also one way to think about the goals of research in basic sensory perception and robotics.

Unfortunately, the basic concepts for which we can establish clear relationships are very far removed from the useful high-level concepts. On the one hand, we have set theory, photons hitting visual receptors, and voltage inputs to motors. On the other hand, we have concepts such as Warcraft, fatigue, spam, people, vegetables, playing, fuzzy, running, and weaving. A central challenge in AI is to bridge this gap.

2 What is FOPC?

FOPC's model of the universe is a set of objects. The universe contains all objects that you intend to reason about. It contains concrete objects such as people and carrots, and also abstract objects such as π, linked lists, and fatigue. Each object has various properties (e.g. a tree may be deciduous). And there are relations between objects (e.g. my cat is on the TV).

FOPC also has the quantifiers "for all" and "there exists." This makes it much more powerful than propositional logic, which is similar but lacks the quantifiers.

In FOPC, variables can refer to objects but not to functions or predicates. This is what makes FOPC "first-order." This limits the expressive power of FOPC, because one cannot express (at least not in the obvious way) generalizations about all functions such as the following definition of function composition.

$$\forall x, y, z, f, g, f(x) = y \wedge g(y) = z \Leftrightarrow g \circ f(x) = z$$

These restrictions can be removed by extending FOPC to produce "second-order" logic. Second-order logic is mathematically well-formed and is the standard language of mathematics. However, less is known about producing efficient computer reasoning algorithms for second-order logic, so it is avoided by AI researchers.

3 Basic FOPC notation

Here are two examples of FOPC sentences:

$$\forall x, \text{carrot(x)} \Rightarrow \text{orange(x)}$$

$$\text{office(off37)} \wedge \text{office(off203)} \wedge \text{resident(hodas, off37)} \wedge \text{resident(hadas, off203)} \wedge \text{adjacent(off37, off203)}$$

The first is a translation of the English sentence "All carrots are orange." The second translates "Hodas and Hadas have adjacent offices."

The concepts in FOPC are very well standardized. Unfortunately, the notation is not. Different authors and subfields have different notation. Moreover, different conventions are used for human-readable logic and datastructures used by algorithms. For example, the following are all possible ways to write the first example above.

$$(\forall x)(\text{carrot(x)} \rightarrow \text{orange(x)})$$

$$\forall x, \mathrm{carrot(x)} \supset \mathrm{orange(x)}$$

(for-all x (implies (carrot x) (orange x)))

An FOPC sentence is made up of constants, variables, function names, predicates, equality, connectives, quantifiers, and parentheses.. A constant refers to a specific object in the universe, e.g. π, Keller, happiness, the set of all oranges. A variable refers to an unspecified object. Each FOPC notation has some convention for distinguishing variables from constants, e.g. variable names may start with a question mark, constants may start with capital letters, or each name must be declared explicitly as a variable or constant.

Functions and predicates are similar to what you might expect. A function takes some objects as input and produces another object. Typical functions include cos, father-of, car, temperature-of, +, and color-of. Notation is usually prefix, but some systems may allow infix notation for arithmetic (e.g. $(x \times y) + z$). Because logic describes a static universe, a function with no input parameters always returns the same value. Therefore, it is not different from a constant.

A predicate is like a function except that it returns a boolean value: true or false. Sample predicates include is-orange $\leq$, owns, and is-illegal. Again, infix notation is sometimes used for common arithmetic predicates, e.g. $x \leq y$.

Equality $=$ is a predicate that receives special treatment e.g. from inference rules. The expression $x = y$ means that x and y refer to the same object in the universe.

Connectives combine two booleans to produce another boolean. The connectives in FOPC are $\vee$ (or), $\wedge$ (and), $\neg$ (not), $\Rightarrow$ (implies), and $\Leftrightarrow$ (if and only if). In most programming languages, it matters which input comes first in AND and OR. This is not true in logic, because logical expressions merely describe and do not perform actions. Also notice that $\vee$ is OR (true if both inputs are true) and not XOR (false if both inputs are true).

BEWARE: In mathematics and formal computer science, $p \Rightarrow q$ is true in two cases. It is true if p is true and q is true (which you probably expected). However, it is also considered true if p is false, regardless of whether q is true or not. This is counterintuitive because it does not match the usage in standard English, which requires some type of a causal connection if two clauses are joined by "implies" (or "if ... then"). Therefore $p \Rightarrow q$ means the same thing as $q \vee \neg p$.

Quantifiers are used to express generalizations about all objects in the universe. There are two quantifiers in FOPC: $\forall$ (for all or for every) and $\exists$ (there exists). The first is called the "universal quantifier" and the second the "existential quantifier." For example:

$$\forall x, \mathrm{age-of(x)} \geq 21 \Rightarrow \mathrm{can-drink(x)}$$

"Everyone over the age of 21 can drink."

$$\exists s, \mathrm{female(s)} \wedge \mathrm{daughter-of(s, Bill)}$$

"There is some woman who is the daughter of Bill."

In some notational systems, absence of an explicit quantifier on a variable means that the variable should be interpreted as universally quantified. I will always use explicit quantifiers in human-readable logic. However, quantifiers will be dropped in the data structures for certain computer algorithms.

Finally, FOPC uses parentheses to indicate how a sentence should be parsed. In human-readable logic, they are only used when the parse seems ambiguous. Computer representations usually use parentheses around all sub-expressions.

4 More about quantifiers

The basic meaning of quantifiers is very similar to that in ordinary English. However, a variety of details may be surprising, either because they differ in mathematical English, because mathematical usage is more precise, and/or because ordinary English avoids using quantifiers in complex ways.

Each quantifier introduces a local variable, called a "bound" variable. Like local variables in computer languages, or the variable in an integral, the name of this variable is irrelevant. The meaning of a sentence stays the same if you (consistently!) rename a bound variable.

In human-readable logic, the scope of a bound variable extends from its quantifier to the end of the sentence. In computer-readable logic, the scope is usually indicated by parentheses. It is possible for a sentence to contain two variables with the same name, as in the following example. However, this is *very bad style* and will lead to tears. Do not do it and prevent your computer programs from doing it.

$$\forall x, f(x) = 3 \wedge \forall x, g(x) > x + \sqrt{2}$$

A sentence may contain no variables (e.g. $\mathrm{married(Bill, Hilary)}$) or only bound variables (e.g. $\forall w, \mathrm{married(Bill, w)}$). In both of these cases, the sentence is either true or false. Which it is depends on the state of the world.

Some sentences contain variables that are not bound by a quantifier ("free variables"), e.g. $\mathrm{married(Bill, w)}$. These are called "open sentences" and they are neither true nor false. You will typically see open sentences as subparts of a larger sentence, or as intermediate stages in building up a larger sentence. An open sentence can be thought of as a statement about its free variables.

When several quantifiers occur in a row, the order matters except when the quantifiers are all the same type (e.g. all universal). For example, the following two sentences mean different things:

$$\forall s, \exists r, \mathrm{student(s)} \wedge \mathrm{room(r)} \wedge \mathrm{occupies(s, r)}$$

$$\exists r, \forall s, \mathrm{student(s)} \wedge \mathrm{room(r)} \wedge \mathrm{occupies(s, r)}$$

To understand the difference, try covering up the first quantifier and figuring out the meaning of the (open) sentence remaining. In the first example, the sentence without its first quantifier is a statement about a student, saying that that student occupies a room. So the full sentence states that every student occupies a room. The room for each student may be different.

In the second example, covering up the first quantifier produces a sentence about a room r. It states that all students occupy the room r. That is, all students occupy *the same* room r. Adding back the first quantifier, the whole sentence means that there is a (single) room which is occupied by all the students. (Ouch, what a crush!)

5 Typed variables

In mathematics, variables are traditionally typed. That is, the universe is divided into sets (e.g. people, vegetables, integers). When a variable is introduced by a quantifier, it is specified as belonging to a certain set. For example

$$\forall x \in \mathbb{R}, x^2 \geq 0.$$

Variables are occasionally introduced and types declared in other ways, e.g. "let $x \in \mathbb{R}$, ..."

Typed variables could be used in AI algorithms, e.g.

$$\forall s \in \text{students}, \exists \text{r} \in \text{roomsoccupies}(\text{s}, \text{r}).$$

Reasoning rules (e.g. unification matching) would then be constrained to match only variables of compatible types. With appropriate care in setting up the details, this would seem like a good thing, e.g. it would prevent sentences from being applied to things they weren't intended to apply to.

However, AI researchers traditionally use untyped variables, though I have yet to uncover a good reason for this practice. Type information is frequently omitted and, when present, is indicated using type predicates:

$$\forall s, \exists r, \text{student}(\text{s}) \wedge \text{room}(\text{r}) \wedge \text{occupies}(\text{s}, \text{r}).$$

6 Set Theory notation

We will need various standard notation from set theory.

Standard sets such as $\mathbb{R}$, $\mathbb{Z}$, $\mathbb{R}^3$. $\emptyset$ is the empty set, i.e. the set with no elements.

We have membership in sets ($x \in \mathbb{R}$) and set inclusion ($A \subseteq B$). Notice that the latter allows A and B to be equal. Also union and intersection of sets $A \cup B$ and $A \cap B$.

A set can be constructed by giving some property that all elements must satisfy, e.g. $\{x | x \in \mathbb{R} \wedge x \geq \pi\}$. This is sometimes also written $\{x : x \in \mathbb{R} \wedge x \geq \pi\}$. Notice that the first notation looks like the textbook's notation for adjoining an element to a set.

Intervals of the real line may be written as in $[3, 47]$, $(3, 102)$, $(-2, \infty)$. Beware: the standard notation for closed intervals looks like the book's notation for lists.

7 Note on set theory

Pages 199-200 of Russell and Norvig present a set of axioms for set theory. Although those axioms aren't wrong, they aren't sophisticated enough to encode all the standard facts about sets. The standard set of axioms, called the Zermelo-Fraenkel axioms, is more complicated. For a readable presentation of them, see Saunders MacLane, Mathematics Form and Function, Springer-Verlag, New York, 1986. (Blais claims the library has two copies.)

8 Defining functions

In using FOPC, one frequently starts with a small set of basic concepts (functions, predicates, concepts) and augments this list by using the basic concepts to define more advanced ones. Mathematicians use the following notation to define a function. Notice that the first line is effectively a type declaration and the second is effectively the code.

$$f : \mathbb{R} \to \mathbb{R}$$
$$f(x) = x^3 + 14$$

The formula for the function is sometimes omitted, if the existence of the function has been deduced in a non-constructive way.

In AI, the type information is frequently omitted and the code is specified using lambda notation (similar to the lambda operator in Scheme). For example, $f = \lambda x, x^3 + 14$. Functions can be specified by formulas (as in this example) or using the full power of computer code. For example, some functions are specified by table lookup: the output values are produced by looking up the input values in a table.

9 Various terminology and notation

A "term" is a constant, variable, or functions applied to constants and variables.

A "ground term" is a term containing no variables.

An "atomic sentence" is a predicate applied to some terms (for some authors: some ground terms).

A "literal" is an atomic sentence or the negation of an atomic sentence.

A "sentence" is anything syntactically correct made from atomic sentences using connectives and quantifiers.

A "well-formed formula" (wff) is a sentence containing no free variables. (Some authors simply call this a "sentence.")

10 Examples of using FOPC in AI

FOPC can be used to formalize the basic properties of predicates. For example, inside and outside are opposites, inside and parallel are transitive, and sibling is reflexive.

$$\forall x, y, z, \mathrm{inside(x,y)} \wedge \mathrm{inside(y,z)} \Rightarrow \mathrm{inside(x,z)}$$

$$\forall x, y, \mathrm{inside(x,y)} \Rightarrow \mathrm{outside(y,x)}$$

$$\forall x, y, \mathrm{inside(x,y)} \Rightarrow \neg\mathrm{outside(x,y)}$$

$$\forall x, y, \mathrm{sibling(x,y)} \Leftrightarrow \mathrm{sibling(y,x)}$$

$$\forall x, y, z, \mathrm{parallel(x,y)} \land \mathrm{parallel(y,z)} \Rightarrow \mathrm{parallel(x,z)}$$

FOPC can be used to define concepts in terms of more basic concepts:

$$\forall x, \mathrm{rational(x)} \Leftrightarrow \exists \mathrm{p, q, integer(p)} \land \mathrm{integer(q)} \land \mathrm{q} \neq 0 \land \mathrm{x} = \frac{\mathrm{p}}{\mathrm{q}}$$

$$\forall x, \mathrm{father(x)} \Leftrightarrow \mathrm{male(x)} \land \exists \mathrm{y}, \mathrm{parent - of(x,y)}$$

$$\exists \mathrm{bag}, \mathrm{empty(bag)} \Leftrightarrow \neg\exists \mathrm{y}, \mathrm{inside(y,bag)}$$

We can express "synchronic rules," i.e. relationships between concepts at a single point in time.

$$\forall x \mathrm{inside(x,freezer)} \Rightarrow \mathrm{cold(x)}$$

$$[(\forall x, \mathrm{missionary(x)} \Rightarrow \mathrm{on(leftbank,x)}) \land (\forall \mathrm{y}, \mathrm{cannibal(y)} \Rightarrow \mathrm{on(leftbank,y)})] \Rightarrow \mathrm{puzzle - solved}$$

$$\forall x, \mathrm{tuna - salad(x)Rightarrowcontains(x,tuna)} \land \mathrm{contains(x,celery)} \land \mathrm{contains(x,mayonaise)}$$

$$\forall x, \mathrm{cat(x)} \Rightarrow \mathrm{furry(x)} \land \mathrm{small(x)}$$

$$\forall x, \mathrm{parallel(x,y)} \Rightarrow \neg\mathrm{intersects(x,y)}$$

We can express the effects of actions. For example, suppose that we are describing the motion of an actor in a computer game, which has a 2D integer grid of locations. We might then describe the effect of moving right as follows:

$$\forall a, x, t, \mathrm{at(a,x,y,t)} \land \mathrm{move - right(a,t)} \Rightarrow \mathrm{at(a,x+1,y,t+1)}$$

The actor can't move through walls. We could express this in a variety of ways, including the following two methods:

$$\forall a, x, y, t, \mathrm{at(a,x,y,t)} \land \mathrm{wall(x+1,y)} \Rightarrow \neg\mathrm{move - right(a,t)}$$

$$\forall a, x, y, t, \mathrm{at(a,x,y,t)} \land \mathrm{wall(x+1,y)} \land \mathrm{move - right(a,t)} \Rightarrow \mathrm{at(a,x,y,t)} \land \mathrm{injured(a,t+1)}$$

For full reasoning about sequences of actions, it is not sufficient to describe how an action changes the world. It is also necessary to describe what properties of the world it does not change (frame axioms), and also what actions could have given rise to certain properties. For example:

$$\forall a, x, y, t, \mathrm{at(a,x,y,t)} \wedge \neg \mathrm{move-right(a,t)} \Rightarrow \mathrm{at(a,x,y,t+1)}$$

$$\forall a, t, \mathrm{pkg}, \mathrm{holding(a,pkg,t)} \Rightarrow \mathrm{holding(a,pkg,t-1)} \vee \mathrm{grabbed(a,pkg,t-1)}$$

A number of these examples illustrate why typed variables, though not traditional in AI, might be useful. The above statements make sweeping generalizations, which are rarely true for randomly chosen objects. Figuring out the correct conditions to put on a generalization is much easier when you can restrict your attention to a limited domain. In FOPC written for a human audience, the variable and function names give you clues about the intended domain (e.g. "bag" or "parallel"). To a computer, however, these are just random symbols.

11 Implies and entails

In addition to a formalism for writing down facts, FOPC provides a variety of reasoning (or inference) rules for deriving facts from other facts. When discussing reasoning rules, we need to distinguish two relations between sentences. A set of sentences $\alpha_1, \ldots, \alpha_n$ "entails" a sentence β, written $\{alpha_i\} \models \beta$, if β is necessarily true whenever all the α_i are true. We write $\{alpha_i\} \vdash \beta$, "*beta* can be inferred from $\{alpha_i\}$," if β can be derived from the α_i using our reasoning algorithm.

Outside of formal logic, "entails" ($\models$) tends to be confused with "implies" ($\Rightarrow$). Also, even when each author makes a clear distinction, one author's symbol for "implies" may be used by a different author for "entails." For example, some pure mathematicians use the symbol *Rightarrow* for "implies" and the English word "implies" for "entails." Or they may use the symbol *Rightarrow* for both concepts.

This confusion happens because the two concepts are very closely related. We use "implies" when writing sentences of FOPC to go in our reasoner's database of facts. It's how the reasoning algorithm writes down generalizations for its own use. We use "entails" when we are looking at the reasoning algorithm from outside, explaining what the reasoner is doing and why.

The distinction between "implies" and "entails" is just like the distinction between the operator "if" used in a piece of computer code and the English word "if" used when discussing what piece of code does. Computer science authors often distinguish these two uses of "if" by writing the two "ifs" in different fonts.

12 Reasoning rules for humans

When humans do pure mathematics, they use a large number of reasoning rules. It is possible to reduce these to a small, minimal set of rules. (And this is one favorite game of logicians and others working on low-level "foundational" mathematics.) However, people find minimal sets of rules harder to use than larger sets, and the resulting proofs harder to read.

Equivalences specify that certain chunks of FOPC can be rewritten, even if they occur as one part of a complex expression.

$$\begin{aligned} \alpha \Rightarrow \beta &\equiv \neg\alpha \vee \beta \\ (\alpha \wedge \beta) \vee \gamma &\equiv (\alpha \vee \gamma) \wedge (\beta \vee \gamma) \end{aligned}$$

$$\begin{aligned}(\alpha \vee \beta) \wedge \gamma &\equiv (\alpha \wedge \gamma) \vee (\beta \wedge \gamma) \\ \alpha \wedge \beta &\equiv \beta \wedge \alpha \\ \alpha \vee \beta &\equiv \beta \vee \alpha\end{aligned}$$

The second and third of these are called distributive laws. The fourth and fifth state that AND and OR are commutative (order of inputs doesn't matter).

There are five equivalences for reasoning about negation (the second and third are known as DeMorgan's Laws).

$$\begin{aligned}\neg\neg\alpha &\equiv \alpha \\ \neg(\alpha \wedge \beta) &\equiv \neg\alpha \vee \neg\beta \\ \neg(\alpha \vee \beta) &\equiv \neg\alpha \wedge \neg\beta \\ \neg(\alpha \Rightarrow \beta) &\equiv \alpha \wedge \neg\beta \\ \neg(\forall x, \alpha) &\equiv \exists x, \neg\alpha \\ \neg(\exists x, \alpha) &\equiv \forall x, \neg\alpha\end{aligned}$$

Other inference rules can only be applied in one direction:

$$\begin{aligned}\alpha_1 \wedge \ldots \wedge \alpha_n &\vdash \alpha_i \\ \alpha_1, \ldots, \alpha_n &\vdash \alpha_1 \wedge \ldots \wedge \alpha_n \\ \alpha_i &\vdash \alpha_1 \vee \ldots \vee \alpha_n \\ \alpha \Rightarrow \beta, \alpha &\vdash \beta \\ \alpha \vee \beta, \neg\beta &\vdash \alpha \\ \alpha \vee \beta, \neg\alpha \vee \gamma &\vdash \beta \vee \gamma \\ \neg\alpha \Rightarrow \beta, \alpha \Rightarrow \gamma &\vdash \beta \vee \gamma\end{aligned}$$

The fourth rule is known as "modus ponens." The fifth is known as "unit resolution" and the sixth as "resolution." The seventh is "proof by cases," a very close relative of resolution.

13 More complicated rules for humans

When sentences contain quantifiers, the notation for writing down reasoning rules gets a big complicated. I will use the notation subst({substitutions}, expression) to express the result of applying a set of substitutions to an expression. For example

$$\mathrm{subst}(\{\mathrm{x/cat}\}, \mathrm{in}(\mathrm{food}(\mathrm{x}), \mathrm{cabinet})) = \mathrm{in}(\mathrm{food}(\mathrm{cat}), \mathrm{cabinet}))$$

$$\mathrm{subst}(\{\mathrm{x/office}, \mathrm{y/parsons-lab}\}, \mathrm{in}(\mathrm{Josh}, \mathrm{x}) \vee \mathrm{in}(\mathrm{Josh}, \mathrm{y}) = \mathrm{in}(\mathrm{Josh}, \mathrm{office}) \vee \mathrm{in}(\mathrm{Josh}, \mathrm{parsons-lab})$$

Then, we have the following rule for deducing specific facts from general statements:

$$\forall x, \alpha \vdash \mathrm{subst}(\{\mathrm{x/val}\}, \alpha)$$

Val can be any ground term, e.g.

$$\forall x, \mathrm{needs(x, caffeine)} \vdash \mathrm{needs(Keller, caffeine)}$$

From a specific example, we can deduce an existential statement:

$$\alpha \vdash \exists x, \mathrm{subst}(\{\mathrm{val/x}\}, \alpha)$$

where val is a ground term that appears in α. For example:

$$\mathrm{needs(Fleck, food)} \vdash \exists \mathrm{x}, \mathrm{needs(x, food)}$$

If x is a variable that does not occur free in α, then we have the following two rules:

$$\begin{array}{lcl} \alpha \Rightarrow \beta(x) & \vdash & \alpha \Rightarrow \forall x, \beta(x) \\ \beta(x) \Rightarrow \alpha & \vdash & \exists x, \beta(x) \Rightarrow \alpha \end{array}$$

These rules formalize a standard sort of proof in mathematical English, which starts with something like "let x be an arbitrary integer," derives the fact P(x), and then asserts that "since x was arbitrary, we have shown that P(x) is true for all x."