Representing Knowledge Symbolically

Much of the knowledge that we can informally express in ordinary language can be expressed symbolically in a way that enables inference

We – unlike other animals – have vast amounts of knowledge *that we can put into words*; and this knowledge makes possible our social interaction through language, our commonsense reasoning about the world, and our planning of immediate and future actions. Our knowledge includes tens of millions of specific facts, about ourselves and our lives, about family, friends and others, our possessions, the things and places in our environment, facts we have learned about history, about geography, politicians and world affairs, fictional items from novels, movies, mythology, etc. Our knowledge also includes comparable amounts of *general* knowledge about the properties, and behavior of people, social groups and organizations, animals and other environmental entities, numerous artifacts (including computers and programs), physics, mathematics, biology, foods, drugs, and other substances, etc.

Drew McDermott argues in *Mind and Mechanism* that human-like consciousness will be achieved in machines only if they acquire human-like cognitive abilities and self-models. Though McDermott is skeptical about a general internal knowledge representation, the very fact of language, and the uniformity of semantic categories across all languages, suggests that we do have such a "Mentalese", capable representing internally whatever we can express overtly in language. Thus it seems that human-level AI systems will also need such a Mentalese, and will have to acquire the same kinds and amounts of knowledge that people possess, if they are going to converse, or reason, or plan intelligently.

Why "logical" representations?

In the following, we will repeatedly see the term "logic" (or "logical") in connection with symbolic representations of knowledge. This is traditional terminology, but it is important not to be misled by it. In particular, using a "logical" symbolism does *not* mean we are committing to building "logical AI systems", i.e., ones capable only of strict deductive reasoning. Quite the contrary: Effective reasoning in the world depends very heavily, perhaps predominantly, on "jumping to conclusions" based on limited evidence. For example, you can't *deduce* that the flight you're taking won't crash, that the server in the restaurant will bring you the food you ordered, that your pet dog will wag its tail rather than bite you when you come home, that you won't get mugged tomorrow, etc.;

yet we live by such presumptive inferences. Our AI systems will need the same ability to make these inferences − even though they should also be able to reason deductively (just as we are).

Using logical representations does not commit us to any particular mode of inference; it merely commits us to a certain (rather language-like) *syntax*, and a *denotational semantics*. The idea of denotational semantics is that we can put symbols into correspondence with a "world", in a way that allows us to formalize the notion of "truth". More specifically, we can decide what the atomic constituents of symbolic expressions refer to, in whatever domain we have chosen, and having done so, we can judge whether a sentential expression is true or false in that domain.

For example, suppose we decide that the symbol *Sn* refers to a certain dog, Snoopy, that the symbol *Tw* refers to a certain bird, Tweety; that the symbol *D* refers to the set of all dogs; that the symbol *B* refers to the set of all birds; and that the symbol *Like* refers to the set of all pairs of entities x, y such that x likes y. Further, suppose the pair of Snoopy and Tweety is in that set of pairs, i.e., Snoopy likes Tweety. Then we can specify what it means for sentential expressions like

D(Sn), D(Tw), B(Sn), B(Tw), Like(Sn,Tw)

to be true or false: *D(Sn)* is true iff (i.e., if and only if) Snoopy (denoted by *Sn*) is in the set of dogs (denoted by *D*); thus by our choices above, it is true. *D(Tw)* is true iff Tweety (denoted by *Tw*) is in the set of birds; thus by our choices above, assuming we consider the set of birds disjoint from the set of dogs, it is false. And *Like(Sn,Tw)* is true iff the pair of Snoopy and Tweety (denoted by *Sn* and *Tw*) is in the set of pairs of entities where the first likes the second; and by our choices above, this is true. This way of defining truth can be generalized to arbitrarily complex sentential expressions, such as

$\neg D(Tw) \wedge \forall x\, (D(x) \Rightarrow L(x, Tw))$,

but we won't do that here (it's CSC 244/444 material). Instead we'll focus on syntax, and rely on an intuitive understanding of the denotational semantics of the symbolism. Such an intuitive understanding relies on some familiarity with math and boolean logic, and more importantly on our understanding of how ordinary language can correspond to entities, properties, and relations in the world. After all, logical representations are derivative from ordinary language...

First-order logic (FOL) as a representation

But if logical representations are derivative from language, and intended to capture knowledge expressible in language, why don't we use ordinary language directly for representing knowledge in a computer? In a word, the problem is *ambiguity*. For instance, imagine trying to tell a computer the following fact:

I saw the moons of Jupiter with a telescope last night.

If we imagine the computer just storing this away as is (as a string or as a sequence of words), we can see that the following problems could arise:

- The computer wouldn't know who "I" is − it may even interpret this as being about itself.
- It wouldn't know what "last night" is − what this means all depends on when it was uttered. Also, it could modify "telescope" (cf., "the accident last night was terrible").
- It wouldn't know whether "with a telescope" modifies "saw" or "moons" (i.e., it might be that some moons of Jupiter have telescopes on them, and those are the ones I saw!)
- The verb "saw" could refer to the action of "sawing"!
- "Jupiter" could refer to the god, rather than the planet

We might say that FOL is a formalized, unambiguous form of ordinary language that avoids problems like the above. Its invention is one of the most important intellectual achievements of the 19th and 20th centuries, and has had an enormous impact on mathematics, philosophy, and AI.

Basically, the way FOL avoids problems like the above is by

- using brackets and fixed operator-operand ordering to avoid syntactic ambiguity;
- allowing any one symbol (like "I" or "saw") to denote just one thing, called its *interpretation*;[1]
- using formalized versions of *and, or, not, implies, is equivalent to,* and *is identical to*, as a means of combining information, where these operators have fixed, well-defined meanings;
- using *quantifiers* $\exists$, $\forall$ and *variables* like x, y, z to talk about an existing, but unspecified individual, or about all individuals.

In the early, somewhat arrogant days of AI, people working on knowledge representation (KR) often thought they could safely ignore logic as a representation, since after all they were working within a completely new "paradigm" − a new way of looking at reasoning and intelligence, namely in terms of *symbolic information processing.* They came up with many supposedly new representations inspired by the computational needs of question-answering, and simple inferencing. But ultimately these representations turned out to be very, very close to FOL, but with a new terminology, and new ways of writing

[1] More generally, logic avoids *context-dependent* meanings, evident in such sentences as *"He got one too"*, or questions like *"What about you?"*.

things down or drawing them. There *were* some genuinely new features, particularly notational devices (and data structures) that indicated how knowledge was to be *accessed and used*; but in terms of *content* of the knowledge representation, the new notations tended to be a re-invention of FOL − but less precise!

In retrospect, this is not really surprising. *Any* representation for particular and general facts about the world surely needs at least the following devices:

- A way of referring to individuals that we want to say something about, and a way of saying that an individual has a certain property, or that certain individuals are related in a particular way (e.g., that Mary is single, or that she is married to John, or that the US borders on Canada);
- Ways of saying that a certain statement is true *and* a certain other statement is true as well; that one *or* another of two statements are true; or that a statement does *not* hold;
- A way of saying that all individuals (of a specified type) have a certain property (e.g., that all birds have wings).
- A way of saying that two things are identical (e.g., that 2 plus 2 equals 4, or that the Morning Star is the planet Venus)
- A way of referring to entities that are functionally determined by other entities (e.g., the *sum* of 2 and 2; the *weight* of an object; the *surface* of an object)

But taken together (and allowing arbitrary depth of embedding of connectives and quantifiers), these requirements virtually force you to adopt (at least) FOL as a representation! In fact, the last requirement is really technically redundant, as is the requirement for "or" ("and" together with "not" can express "A or B", viz., "not ((not A) and (not B))").

So, better to learn it than reinvent it! A lot is known about how to precisely characterize *meaning* in FOL, and how to perform inferences (see below), and it is both very difficult and a waste of time to keep rediscovering these things, in slightly different notations. That is not to say "FOL is all you need"; it isn't. But it's an important, fundamental part of what you need!

The syntax of first-order logic

The list of requirements above very directly motivates the syntax of FOL. The syntax allows us to form formulas similar to English sentences, Above we used symbols like *Sn* for a certain dog, Snoopy, symbol *D* for the set of dogs, etc. This was intended to make clear two distinctions: the distinction between symbols and real-world entities like particular dogs or birds or sets of these; and, the distinction between natural language symbols and FOL symbols. For example, *Like* as an FOL predicate doesn't *have* to denote

the set of entities where the first likes the other − it can mean whatever we want it to mean! That said, in work on KR we usually find it convenient and natural to "echo" the words of ordinary language in the basic symbols that we employ. This makes it easier to remember what we *intended* the symbols to mean. Thus we might write sentential expressions ("formulas") such as Dog(Odie) ("Odie is a dog"), or Loves(Romeo,Juliet), or (∀x (Dog(x) ⇒ ¬Can-talk(x))) ("For any object x, if x is a dog, it cannot talk", i.e., "Dogs can't talk"). We build such formulas systematically from smaller pieces, as follows.

Terms

First, the *basic terms* used to refer to individuals are

individual constants: A, B, C, John, Mary, CSC191, Block1, Block2, ...[2]

and (here we use lower case)

individual variables: x, y, z, x1, y1, block1, block2, ...

(Since there are no other variables in FOL, we often just say "variables".) Two points should be noted about the interpretation of individual constants (informally speaking). As already noted, we can let them refer to *any* individuals we please. So *A* or *John* (or both) could refer to John or to the US or to the number 17 or to the chair you are sitting on, etc. (assuming that these are things you wish to be able to talk about in your knowledge base). In this respect logic is unlike a natural language − in English we cannot usually use *John* to refer to countries, numbers, or chairs, except perhaps by special prior agreement, as some sort of secret code or joke. Another point is that since in logic meanings are unambiguous, a constant denotes exactly one thing.[3] This is also unlike ordinary language, since a name like *John* can refer to various individuals with that name, depending on context.

It can also be quite useful to allow numerals

0, 1, 2, ..., 10, 11, 12, ...

as individual constants. Given the above "freedom of interpretation", numerals could refer to people or countries, or anything else; however, it is more common to constrain the interpretation of numerals so that it is consistent to regard them as denoting numbers.

For referring to the values of functions, we introduce lower-case

function constants: f, g, h, weight, father-of, surface-of, sum, ...

For each function constant, we have to decide how many arguments it takes − this is called its *arity* (or *adicity*). Thus *weight* might be a unary (or, monadic) function constant (arity 1), and *sum* might be a binary (or, dyadic) function constant (arity 2). But note that function constants can be interpreted to mean anything we want them to mean, just as in

[2]We use capitalized identifiers here, although conventions vary.

[3]For a quantified variable, we "iterate" over its alternative values

the case of individual constants. For instance, *weight* might actually refer to the surface area of an object, and *sum* to the distance between two given objects; however, we usually do not choose our constants so perversely! Using functions, we can form

function terms: weight(John), sum(weight(x),weight(y)), ...[4]

In general, we can recursively define **terms** as being either individual constants, or individual variables, or functions applied to an appropriate number of terms. Terms containing no variables are called **ground terms**, while others are **non-ground terms**.

An important technical point here is that we assume that functions always have values, no matter what objects they are applied to. (They are *total* functions.) For example,

weight(17), father-of(Pacific-Ocean)

are presumed to have values, perhaps chosen in some arbitrary way since we don't really expect to refer to these values.

In addition to ordinary function constants like those above, some versions of FOL also allow certain mathematical function symbols, such as

$., +, -, *, /, \uparrow, \cup, \cap.$

These constants have certain "intended interpretations", and can be written in infix rather than prefix form; e.g.,

(A . x), (A + 2), etc.,

rather than .(A,x), +(A,2), etc. (".” is intended to refer to addition of an initial element to a sequence, like the Lisp *cons* function.) However, unless we somehow define or constrain the interpretations of these function constants, they can in principle refer to any functions at all.

Formulas

Formulas (also often called *sentences*) describe properties and relationships (including identity) of objects. Thus, besides terms referring to objects, we also need symbols referring to properties and relations. These are called

predicate constants: A, B, C, Dog, Person, Loves, Married-to, Smokes, ...

These again have to have a fixed arity, such as 1 in the case of *Dog* and 2 in the case of *Loves*. And again these predicate constants can refer to any properties and relationships we like, not necessarily the ones we would expect from their resemblance to English.

Note that we allow any capitalized (or completely upper-case) symbols for both individual constants and predicate constants. However, any one symbol can be used in only

[4]The case conventions used here fall by the wayside when we write formulas in Lisp, since Lisp is case-insensitive; also in Lisp we would put the function name inside the brackets, e.g., (WEIGHT JOHN), rather than weight(John).

one way. Similarly a lower-case symbol can be used as a variable or function constant, but not simultaneously for both. Also note that we need not explicitly specify whether something is an individual constant, function, or predicate − it can be inferred from the way we *use* these symbols (if, in fact, we do use them consistently). Similarly, the arity of functions and predicates is evident from the way we use them.

We should note here that we also allow the *equality predicate*, =, but rather than being freely interpretable, this predicate has a fixed meaning: it holds only for a pair of arguments that refer to the same object. As in the case of binary mathematical functions, we use infix form (possibly surrounded by brackets, if there is ambiguity)

A = B, etc.,

for equality statements, rather than prefix form =(A,B), etc. The presence of the equality predicate distinguishes FOL from the *First-Order Predicate Calculus* (FOPC), i.e., FOL is FOPC plus equality. Certain additional mathematical predicates may be useful as well, such as

$<,\ >,\ \leq,\ \geq,\ \in,\ \subset,\ \supset,\ \subseteq,\ \supseteq$.

Using predicates and terms, we can now form

atomic formulas: Dog(Fido), Loves(Romeo,Juliet), f(x) = y, etc.

Formulas containing no variables are called **ground formulas**.

There are two general ways of forming complex formulas from atomic formulas, namely, by use of logical connectives and by use of quantifiers. We use the following

logical connectives: $\neg,\ \wedge,\ \vee,\ \Rightarrow,\ \Leftarrow,\ \Leftrightarrow$.

Intuitively these mean *not, and, or, implies, implied by,* and *if and only if* (*equivalent to*) respectively. "$\Rightarrow$" may also be written as "$\supset$" and "$\Leftrightarrow$" as "$\equiv$".[5] $\neg$ is a unary (1-place) logical connective, while the others are binary (2-place) logical connectives, and as such allow formation of compound sentences such as

$\neg$Likes(Biden,Putin), Loves(Romeo,Juliet)$\wedge$ Loves(Juliet,Romeo),
At-home(Mary)$\vee$ At-work(Mary), Poodle(Fifi)$\Rightarrow$ Dog(Fifi),
(A > B)$\Leftrightarrow$ (B < A).

Finally, we introduce the two

quantifiers: $\forall,\ \exists$.

Their use is illustrated by the following examples:

($\forall$x Entity(x)), ($\forall$x (Poodle(x) $\Rightarrow$ Dog(x))),
($\exists$x Robot(x)), ($\exists$x (Robot(x) $\wedge$ Smart(x))),
($\forall$x (Robot(x) $\Rightarrow$ ($\exists$y Built(y,x)))),

[5] We prefer "$\Rightarrow$" and "$\Leftrightarrow$" because they are ascii-printable and can be Lisp atoms; also "$\supset$" can be confused with "superset", if we want to use such a relation.

(∃x (Robot(x) ∧ (∀y (Robot(y) ⇒ x=y)))).

These can be read respectively as *Everything is an entity*; *Every poodle is a dog*; *There exists a robot*; *Some robot is smart* (Note that this requires ∧ rather than ⇒ !); *For every robot, there is someone (or something) that built this robot*; and, *There is only one (i.e., exactly one) robot.*

As you see, a quantifier is always followed immediately by a variable, and is said to *bind* that variable. The quantifier and variable are immediately followed by a formula, called the *scope* of the quantifier. The quantifier, variable, and scope are enclosed in parentheses, except where no ambiguity can arise.

There is another way to formalize quantification, which is still a little closer to natural language. In that alternative approach, *Some robot is smart* and *All robots are smart* would be written respectively as

(∃x: Robot(x) Smart(x)),
(∀x: Robot(x) Smart(x)),

i.e., "Some x such that x is a robot is smart", and "Every x such that x is a robot is smart". This is called *restricted quantification*, and it is equivalent, for quantifiers '∃' and '∧ ', to the original versions above, which used connectives '∧ ' and '⇒ '. The restrictor-based approach has the advantage of being generalizable to such natural-language quantifiers as *most, few, almost all, no, etc.*. Except for *no*, these can't readily be expressed in FOL (in fact not at all, in a certain technical sense).

An occurrence of a variable ν in a formula is said to be a **free** occurrence if it is not in the scope of any ∀- or ∃-quantifier that binds ν. A formula that contains no free variables is called a **closed formula**. The term **sentence** is also frequently reserved for closed formulas (though as noted it is commonly used for arbitrary formulas as well). Note that *ground formulas* are always closed, but some closed formulas are not ground formulas, because they contain quantifiers.

BNF notation; first-order logic and first-order languages

For those who have some acquaintance with programming language theory, we can summarize and semi-formalize these syntactic devices as follows, using "BNF" notation (ignoring the infixed, mathematical functions and relations, and also ignoring correct correspondence between the arity of function and predicate constants, and the number of arguments to which they are applied):

⟨**individual constant**⟩ ::= a | A | b | B | c | C | John | Block1 | Block2 | ...
⟨**variable**⟩ ::= x | y | z | x1 | x2 | block1 | ...
⟨**function constant**⟩ ::= f | g | h | weight | sum | mother-of | ...
⟨**term**⟩ ::= ⟨**individual constant**⟩ | ⟨**variable**⟩ |
⟨**function constant**⟩ (⟨**term**⟩ , ..., ⟨**term**⟩)

⟨**predicate constant**⟩ ::= A | B | C | Dog | Loves | Owes | ...
⟨**binary connective**⟩ ::= ∧ | ∨ | ⇒ | ⇐ | ⇔
⟨**formula**⟩ ::= ⟨**predicate constant**⟩ (⟨**term**⟩ , ..., ⟨**term**⟩) | (⟨**term**⟩ = ⟨**term**⟩)
| ¬⟨**formula**⟩ | (⟨**formula**⟩ ⟨**binary connective**⟩ ⟨**formula**⟩)
| (∀⟨**variable**⟩ ⟨**formula**⟩) | (∃⟨**variable**⟩ ⟨**formula**⟩)

Note that expressions of form ⟨**....**⟩ above are *metalinguistic* symbols − they vary over expressions in the formalism being defined, rather than being part of that formalism.

Also, we are still not being perfectly precise: function and predicate constants should really be sorted into 1-place, 2-place, 3-place, ... constants (also called monadic, dyadic, triadic, ..., or unary, binary, ternary, ...). Furthermore, a given alphanumeric string may only be used as one type of constant, with a fixed adicity (arity). In other words, the sets comprising the individual constants, the 1-place function constants, the 2-place function constants, ..., the 1-place predicate constants, the 2-place predicate constants, etc., are *disjoint sets*.

As a final point, observe that depending on exactly what individual constants, function constants and predicate constants we choose, we can get various **first-order languages** using the above schema. For instance, we might limit these constants to some finite sets, rather than having an unlimited supply of them. So FOL, as such, is not itself a unique *language*, but rather it is a *formalism* that allows for many first-order languages.

What FOL can and can't do

We've indicated that logical representations provide an unambiguous way of capturing *factual* (or at least *declarative*) knowledge, of the sort we can readily express in words. But of course storing away factual knowledge is not an end in itself − the question is what we can *do* with such knowledge. In general, what we do besides retrieving relevant knowledge (e.g., to answer a question) is to make *inferences*. While in a full exposition of knowledge representation our next major task would be to spell out in detail the semantics (meaning) of FOL, here we'll just take a quick look at the *use* of FOL to perform inference. (Some would say *reasoning*, though to this term tends to suggest figuring something out by a complex series of inferences.)

Factual knowledge and types of inference

When logicians talk about FOL, they generally have in mind the above syntax (perhaps leaving out some inessential features), a formal semantics of the type hinted at above, and the "deductive proof theory". The deductive proof theory concerns the rules by which we derive true conclusions from true premises. However, let me emphasize again that it would be a mistake to think that in using FOL we are somehow restricted to using deduction for inference (a mistake typically made by people who argue against the use

of logical representations!) As was apparent from the earlier examples about flights not crashing, servers bringing the requested food, etc., there are other important forms of inference besides deduction that FOL lends itself to, and it is worth looking at a quick list of these:

- *Deduction*: deriving logically necessary consequences
  - Mary has a poodle named Fifi. Therefore (given that all poodles are dogs), Mary has a dog.
  - Fifi's mother is Lulu. Therefore (given that offspring are younger than their parents, and that mothers are parents), Fifi is younger than Lulu.
  - Fifi ate a cookie. Therefore (given certain facts about eating, which you can fill in), the cookie went inside Fifi.

- *Uncertain and nonmonotonic inference*: deriving likely consequences in the absence of information to the contrary
  - Given (only) that Tweety is a canary, tentatively conclude that Tweety flies.
  - Given that John is sniffling and has a sore throat but no fever, tentatively conclude that he has a cold. (In either example, further evidence may reverse the conclusion – that's what "nonmonotonic" means.)
  - Given that John and Mary are sitting in a restaurant at a table with some dirty dishes and coffee cups on it, not eating or drinking, conclude that they came into the restaurant (perhaps 1/2 hour to 1 1/2 hours ago), were seated at that table by a host, ordered a meal from a server, ate most of the meal, and either have just paid or are waiting to pay for the meal, shortly after which they will exit the restaurant. (This is a *script-based* inference.)

- *Abduction* (and *induction*): formulating general hypotheses to account for multiple particular observations or facts; (induction: confirming or disconfirming such hypotheses based on observations)
  - Given that all the crows I've seen are black, conjecture that all crows are black. (Induction: gradually confirm the hypothesis after seeing more and more examples, and no counterexamples.)
  - Upon noticing that it is possible to "fit" ellipses to observations of the motions of several planets, conjecture that all planets follow elliptical orbits (Kepler).

- *Explanation*: postulating particular facts which, when combined with other available knowledge, can account for new observed facts (this is sometimes counted as a special case of abduction; it also shades over into uncertain or nonmonotonic inference)
  - Given that my parked car has disappeared, conclude that it was stolen or towed.

  - Given the observation of a slight, periodic wobble in the position of a certain star, conclude that a large planet is orbiting it.

- *Planning and plan recognition*: formulating plans to achieve given goals; and conjecturing other agents' plans based on their actions or stated desires
  - Given my goal of, say, being in Baltimore in early November, I decide to book a flight, arrange for a TA to sub for me in CSC 244/444, etc.
  - As I am about to pull out of my parking spot at a busy supermarket, I see another car stop and wait behind me. I infer that the driver wants to park in my spot, and (probably) go to the supermarket.

The limits of FOL: Analogue representations and "know-how"

Does the above list of inference modes cover the full range of thinking that people are capable of? It seems clear that the answer is "no": We possess both *analogue-like* methods of making inferences, and various specialized skills and know-how.

Analogical inference, then and now. In classical studies of analogical inference (e.g., see Thomas Evans' 1964 ANALOGY program, http://logical.ai/auai/p327-evans.pdf), the goal was to match one geometric figure to another, among several possibilities, so that the relationship between the two figures would be the same as the relationship between two given figures. There were also programs for predicting the next number in a sequence such as 1, 4, 9, 16, ... But it was not appreciated at the time that human analogical thinking might be far more pervasive than such constructed problems from IQ tests might indicate.

The last twenty years or so have made clear that powerful kinds of analogical inference are possible with *vector space* representations of entities and relationships. We can represent many types of objects, including images and words (or larger text segements) in terms of arrays of *feature vectors*. For example, a vector representation of an image, say, of a dog, might contain local features corresponding pixel values (intensity and color) or, at a subsequent processing level in a neural-net-like architecture, edge elements, color patches or texture elements or, at still later processing levels, features corresponding to curves, surfaces, object parts, etc., until at the top level we have a vector whose direction in vector space corresponds to a particular type of object such as a dog or a fire hydrant or bicycle, etc. Now, this way of describing feature vectors in a neural net (NN) is a bit deceptive. In reality, the feature vectors in a NN are *learned* (except at the input level), based on seeing many appropriately *labeled* examples of a concept (e.g., many images of dogs, fire hydrants and bicycles), and in general it's only a supposition that the vector representations at the various "hidden" levels might reflect features that seem intuitively relevant to recognizing an image as an image of a dog, etc. Essentially NNs are "black

boxes" whose internal vector epresentions somehow derive object categories; and close examination of NNs has shown that they may use entirely unexpected "clues" in an image to obtain a correct result.

How is such processing analogical? Well, it turn out that when you've trained a NN on some large set of examples of a concept, it will be able to recognize *new* examples of the same concept, such as correctly identifying an image of a dog as such, though it was never seen before. In a sense then, the NN has made an analogy between previously seen dog images and the new one.

This sort of analogizing is even more impressive when the output of an NN is not just a catageory, but another complex pattern. Suppose, for example, that we provide many examples of "premise sentences" to a NN, along with desired "conclusions". Just for illustration, suppose the example pairs are something like the following:

A woman fell in love => The woman was happy;

A man had an accident => The man was unhappy;

A child received a present => The child was delighted;

A student was mugged => The student was furious;

A man tripped and fell => The man was embarrassed;

etc., i.e., certain eventualities involving some person elicit certain emotions. Such examples could be hand-constructed, but for scaling up, they would be crowd-sourced, or better, similar kinds of pairs would be derived somehow from the vast online textual resources. A NN trained on such data might well learn to correctly produce output sentences similar to the sentences on the right above, corresponding to new input sentences like those on the left. This amounts to a way of inferring human emotions in given situations by analogy with previously seen examples. Note that this type of inference is not just a matter of learning to predict the right emotion, but also learning to express the conclusion in understandable English! The same kind of learning is of course involved in learning to translate sentences of one language into semantically equivalent sentences in another language, which systems like Google Translate do remarkably well.

Does human analogical inference work this way? Well, quite possibly to some extent, especially in some domains (e.g., learning to recognize words in a speech stream). But note the striking difference between classical analogical inference and NN-based inference: The classical tasks are apt to be solvable by people based on a single example or a few examples, just using their general knowledge about geometrical figures, numbers, people, sentence meanings, etc. Note that *you* could solve new examples of the above types, such as the emotion inference examples, just based on the five examples given; and people can solve geometric analogy or sequence extrapolation problems even when they have seen no such problems before. By contrast, NN-based inference is typically based on thousands, millions, or even billions of training examples! Moreover, people can explain *why* they reached an analogical conclusion, such as the kind of transformation involved in mapping one geometrical figure into another, or the mathematics involved in a sequence extrapolation task; whereas NNs are unable to explain their answers, because they were

reached through vector processing, not any knowledge-based reasoning.

Imagistic reasoning. The most "vivid" analogue method people make use of is *imagistic reasoning*. For example, consider the follwoing two scenarios:

a. *"Walking alongside the lioness, Joy Adamson stroked its head".*

b. *"Walking alongside the giraffe, Joy Adamson stroked its head".*

The second scenario strikes us as highly implausible, because when we try to visualize it, we "see" that Joy Adamson wouldn't be able to reach the head of the much taller giraffe. (We might "force" an understanding by imagining the giraffe to be "bowing down" its head, or to be a new-born.) Though one could concoct FOL axioms (about the sizes of various, creatures, reaching, etc.) that would allow an AI system to detect the anomaly by symbolic reasoning, this would be a never-ending knowledge engineering task. Imagine that either of the above sentences were followed by, *"Then she climbed on its back"*. In the case of the giraffe, we are likely to visualize a failed attempt to climb up – again something very hard to capture axiomatically; or we might imagine the intrepid naturalist as possessing incredible climbing skills, or using a ladder, which would be even more challenging to capture in symbols. A much more promising approach is to make use of computer graphics techniques, allowing 3-D dynamic simulation of such scenarios based on models of various familiar physical entities. The goal would be to match human visualization capabilities, which are well-established, e.g., through the work of George A. Miller and (later) Stephen Kosslyn.

Temporal and taxonomic reasoning. Using mental imagery or visualization is not the only special analogue capability we possess. We also absorb large complexes of temporal relationships (such as those among the events in a novel – or in our own lives) with ease, and can determine before/after relationships and approximate durations more or less instantly. In computers, we can use graphs that "mimic" temporal relations and allow rapid relation and duration inference through suitable numeric annotations. Another innate human capability is the classifications of the types of entities we learn about in the world into hierarchies, where we have very general types (such as "physical object", "physical substance", or "abstract entity") at the top, and more specialized types further down, such as a division of physical objects into living and non-living things, where living things are in turn divided into humans, nonhuman creatures, and plants, and so on. This type of hierarchic or taxonomic organization can also be matched in computers through tree-like, labeled data structures allowing efficient inference of relationships among entity types, such as subordination of one type by another (squirrels are living things) or incompatibility of types (squirrels are not monkeys). A case can be made for other analogue reasoning capabilities as well, but we've sufficiently clarified what is at issue here.

Skills (know-how). While the above analogue methods support drawing commonsense conclusions about the world, another important aspect of our cognitive functioning in the world is our ability to acquire subtle know-how or *skills*, including many motor skills and sensory processing abilities. These appear to have nothing to do with logic; rather, they are best viewed as procedures operating on specialized data structures and I/O

streams, and perhaps running on very specialized computational architectures. Imagine trying to implement any of the following activities in a humanoid robot using logical representations and operations as a basis:

- whistling (using a human-like vocal tract and musculature)
- riding a bike, catching a ball, using chopsticks, tying shoelaces
- interpreting retinal images
- speaking grammatically (well, this example may be debatable...)
- learning a new concept, a new language, watercolor painting, ...

Even if we could come up with logical descriptions of how we do these things (or how a humanoid robot could do them), such descriptions probably could not be used in any direct way to implement them.

It would be wrong to conclude, though, that there is a sharp division between knowledge of how to do things − *procedural* knowledge − and factual knowledge. (This has tended to be the view taken by both sides in the old "proceduralist-declarativist controversy" − a controversy that started in the late 60's, with one side defending procedural representations and the other logical representations of knowledge.) For instance, consider the following kinds of procedural knowledge:

- cooking recipes
- assembly and installation instructions (for furniture, electronic equipment, etc.)
- emergency procedures (on airplanes, in medical emergencies, etc.)

etc. These are just as natural to express in language − and FOL − as facts about food, furniture, airplanes, etc. The difference is really just one of "mood": declarative (or "indicative") *vs.* imperative. Declarative sentences express facts while imperative sentences express what to do, but apart from that, they use the same grammatical constituents (noun phrases, verb phrases, etc.), with the same meanings.

In fact in a broad sense, standard programming languages such as Python or C++ can be viewed as logics, but ones that make (mostly) imperative rather than declarative statements.[6] Why, then, is it that programs in most programming languages look so little like FOL? Well, there are several reasons. First, the objects and operations they talk about tend to be ones inside the computer (storage locations, variables, assignment) or abstract mathematical ones (numbers, symbolic objects, subroutines), instead of being real-world things like food and cooking and furniture and so on. Second, the concern in procedural languages is very much with sequencing in time, while FOL makes no special

[6]Prolog gives the illusion of being declarative, but is still interpreted imperatively, i.e., there is an "understood" way of executing a prolog program.

provision for this (any more than for, say, sequencing in a spatial dimension − though it is entirely possible to "talk about" time and space). Third (a point related to the second), they use syntactic devices not seen in FOL − e.g., *begin−end* blocks, *if−then* statements, procedure declarations, loops, lambda-abstraction, etc. And at the same time, they tend not to have quantifiers $\forall$ or $\exists$. (Some languages allow commands of form, "Do such-and-such an operation for *all* objects in a certain set"; but usually not "Do such-and-such an operation for *some* object in a certain set", as this would be rather vague – more precisely, *nondeterministic*.) Still, the meaning of programs can be studied in much the same way as the meaning of declarative logics.[7] As well, certain languages have been developed, called *dynamic logics* and *executable temporal logics*, which have a more FOL-like syntax and can be used as high-level programming languages (or to describe what programs written in other programming languages do); e.g., see H. Barringer, M. Fisher, D. Gabbay, R. Owens and M. Reynolds, *The Imperative Future: Principles of Executable Temporal Logic*, Research Studies Press LTd., 1996.

Expressivity limitations of FOL. As a final comment, it should be noted that FOL is not quite expressive enough to cover everything we can easily express in ordinary language. For instance, it lacks good means for expressing temporal modifiers (as in "John woke up repeatedly during the night"), or for expressing belief (as in "Mary believes that John is an insomniac"). Adding beliefs, wants, desires, and so on is possible in so-called *modal* logics, though the ones logicians have studied are mostly too restrictive, and too focused on deduction, to be suitable for general, human-oriented KR and language understanding. That's a significant point in considering self-models and models (theories) of other agents, since these require attribution of beliefs, intentions, desires, etc., to oneself and others, and as such are crucial for human-like consciousness.

In summary, FOL provides means of symbolically representing much of the knowledge that people can express in language and use for deliberate *thinking* of various sorts. To ultimately endow machines with human-like cognitive abilities, we will need to integrate such symbolic methods with several types of analogical and procedural methods.

[7] E.g., see E. Stoy's book *Denotational Semantics*.