MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence, Fall 2001
Recitation 7, October 26, 2001

Language and Mind Prof. Robert C. Berwick

Agenda

1. **Administrivia**
2. **Language: Why language is special; from Words to Meaning**
3. **Syntactic nets to semantic nets**
4. **The Blue Room: The Language of Thought?**

1. Administrivia: PS due Oct. 30

2. Language: Why language is special; from Word strings to Meaning (Syntax to semantics)

Language is special: The Twain Test (Unsupervised learning; very small sample complexity – 1-5 examples; no Wall Street Journal subscriptions), and "the Burst Effect". At age 1 yr, 1-11 months or so, these are the kinds of utterances children produce:

Ride papa's neck; this my rock-baby; mama forget this

By the age of 3, this is what they produce. What's the difference?

You match me open sandbox; Papa, you like this song? I won't cry if mama wash my hair

The same difference shows up in this way.
Pop-quiz (multiple choice): who produced the following 'sentences' (Names changed to protect the innocent):

(1) I see red one (2) P. want drink (3) P. open door (4) P. tickle S. (5) I go beach (6) P. forget this (7) P said no (8) P. want out (9) You play chicken

Multiple choice: (a) Pidgin speakers; (b) apes (signing); (c) Feral child Genie; (d) ordinary children [Hint: recall the Burst effect plus the general rule about tricky 6.034 questions]

Question: How can we *represent* knowledge of language? How can we *compute* with it? (Like data structure + algorithm question).

Answer to first part (representation) – this is what linguists do for a living. What are the *underlying representations* that constitute our knowledge of language?

Answer to the second part (computation) – this is what syntactic and semantic transition trees partially accomplish, mapping strings of words into procedural operations on a database ("meaning").

As to the first part, look at how much we implicitly 'know' about a language. Take even a simple sentence like *the cats ate ice-cream.*

- We know how each word sounds and whether it's an English word at all. *Ca* begins a valid English word, but no English word will start out *ptk* (but could in Polish). Further, the *s* on *cats* marks it as plural.
- We know that the words must appear in a certain order. *Ice-cream ate the cats* means something very different from *The cats ate the ice-cream.*
- We know "who did what to whom," a kind of mental snapshot: *the cats* is the agent of the action *ate* while *the ice-cream* is the thing eaten, or the affected object. (These are the *thematic roles* or *slots in a thematic role frame*) . And so on…

It is conventional to divide the study of the representation of linguistic knowledge into two parts: syntax, the study of word arrangements without regard for meaning – this derived from the Greek word *syntaxis*, literally 'to arrange together' and semantics, the study of meaning.

Syntax determines the allowable combinations of primitives in a language, where one notion of primitives are word categories, and further, combinations of word categories. For example, let us suppose that we had sentences like these:

The cat ate the ice-cream
The dog ate the ice-cream
The dog ate the cat

If we take these words for the moment as primitives, we can simply write down these possible linear patterns as a *directed graph* or *transition network* (*tree*) as follows:

Here, the 'start' state is marked by an incoming arrow, and the final (goal) state is a double circle. Given a transition network and a string of words (a sentence), we can *check* to see if the string of words can be 'let through' the network, or *generated* by it. To do this, we start by looking at the first word in the sentence can let us move from one state to the next via a *transition* – does the label on the arc match the current word in the sentence we are looking at? (See how we can march through the first example sentence this way. If we arrive at the end of the sentence and are in the goal state, we win.)

Of course, it would get very boring (and wrong) to just keep writing down words as arc labels to encompass more and more sentence patterns: *a dog ate the ice-cream; a cat ate the ice-cream; a dog ate the cat; the dog ate an ice-cream;* etc. Instead, we can *collapse* collections of words such as {dog, cat, ice-cream, tree,…} into a *word category* called 'Nouns' and words like {a, the, an…} into a category called 'Determiners' (because they 'determine' in some sense the multiplicity of the Noun that comes later). That gives us the following, more compact network:

Of course, now this transition tree (network) will allow some 'funny' word sequences – but we did say this was *syntax*, not semantics. Only form matters.

Finally, since we are good programmers, we notice a final redundancy in the network: the pattern "determiner noun" is repeated, once as the 'Subject' of the sentence, and once as the 'Object'. So, why not turn that pattern into a subroutine? We can call the repeated pattern a 'Noun Phrase' and splice out the repeated pattern (see the network below.) Of course, since this is a subroutine, we must have a way to *call* and *return* from subroutines generally (i.e., we must use a *stack* and sentences are now *recursive*). Now we will have two networks: the main Sentence network and a 'subnetwork' called a Noun Phrase. We need a new label in the Sentence network that is called 'Noun Phrase', and marking that at the beginning of a valid sentence, we must find a Noun Phrase:

We now have that in English, a Sentence 'is a' Noun Phrase followed by a Verb and then a Noun Phrase. (This of course is not always so!) We can further combine a Verb and a Noun phrase into a unit called a Verb Phrase. This is what a syntactic transition network can easily encode. We just replace 'isa' and map the 'followed by' with a graphical representation:

How would we modify this to handle an example like "John slept"?

So far, our *syntactic* networks merely check the *form* of sentences. They don't *do* anything. We can couple form (syntax) to actions by adding procedures that trigger as each arc is traversed or as we hit the end of a network (or both). Such actions *translate* the input sentence into a series of actions, often a series of database calls, just as a compiler will parse a line of program code and output the actual sequence of machine instructions to be followed. In the case of computer code, the "meaning" simply *is* the sequence of machine instructions; in the case of a simple natural language system, the "meaning" is the database calls; in general, the "meaning" of a sentence is much more complicated than that. As a simple example, though, following the Winston notes, consider a sentence like "How many screwdrivers are there?" The goal of the network is to map the query into a set of database and procedure calls. As we traverse the portion of the network corresponding to "How many", we output "Count" (a procedure call). As we traverse "screwdrivers are there" we output a database call that 'selects' the screwdrivers and returns a list of them (or, if we are using a real database program, most likely the count routine would be built in and we could do it all at once.) The key idea, though is this:

Distinct paths through the network represent distinct meanings= distinct procedure and database sequence calls

Example: Fruit flies like a banana. There are at least 2 obvious paths (in an expanded network) hence 2 meanings: