MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.034 Artificial Intelligence, Fall 2003
Recitation 8, October 23rd & 24th , 2003

How to do things with Words Prof. Robert C. Berwick

Agenda

1. **Jumping the nets (everything you ever wanted to know about semantic transition trees, but were afraid to ask)**
2. **Transition frames, trajectory frames: multiple representations for language**
3. **Is any of this stuff real?**

1. Jumping the nets. A semantic transition tree (STT) is simply a linear branching pattern, perhaps recursive, that spells out possible linear sequences of word classes, where the classes are based on semantic properties like size, weight, type of object (book, weapon,…), etc. We add 'semantic translation' or 'meaning processing' by attaching a semantic action routine to the end of the linear pattern. Thus each pattern can be thought of in the form of a pair `(syntactic-pattern, semantic-translation)` and what the STT does is to map or translate a string in some input language, say English, into some output language, say, a database query; a transition frame; or anything at all.

   To build an STT suppose for instance that one had to write a pattern for examples such as, "What size is the M16"; "What size is the uzi"; "what type is an uzi'; What type is the m16". These can be abstracted into a single network that looks like the following. We leave constant words alone, they must be detected exactly as they appear in the input; we **combine** strings of words that appear in the same position (the books, weapon size, …) into **classes** (here denoted by brackets):

   What {size | type } **is** {the|an} {m16 | uzi}

   Next we **replace** the brackets with **subroutine calls** to other STTs, standing for the classes:

   What ?noun-group1 is ?noun-group2

   We must build an STT for **noun-group1.** (It is the same as ?noun-group2, but because of inelegance in the STT implementation for Pset3, we have to distinguish it – its role, hence its semantic translation, must be different.) Let's try this (exercise).

   Finally, we must **add** a **semantic translation** or **output mapping** at the end of each pattern, in this case, a piece of a complete db query For instance 'the size' is mapped to `'(size = ,w-size)`, where `w-size` is the evaluation of the actual value pulled from a db, while 'the uzi' is mapped to `` `(weapons = ,?weapons) `` where `?weapons` will evaluate to 'uzi'. The output translation pastes both of these together as returned from the subroutine calls to form the complete query, e.g., `(get db (w-size uzi))`.

   Since any output mapping can be produced, such a system can in general map any strings (describable by the STT formalism) to any other strings. The 3 next problems in the Pset are aimed to get you to realize this, and have you gain familiarity with a more recent and justifiable representational system for language 'meaning' called **event structure.** Event structure is in fact made up of several different subrepresentations: we cannot describe what a sentence 'means' by a single, unified representation. First, the event is centered around an action – the verb (is this always right?). So we might start out by writing an event representation for 'Mary killed the spider' as (kill… But there is more to it. The event structure includes information about causal relations, motion (physical or not), and actors in the action. In short, it must include enough info to answer questions, make inferences, etc. So we need at least:

   1. One kind of event subrepresentation is like a 'mental snapshot' picture – who did what to whom. Winston calls this a **thematic representation.** Here you identify the main 'players' in a sentence – in the 'kill' sentence, Mary and the spider. 'Mary' is the **agent,** while 'spider' is the **affected object** or **beneficiary.**
   2. A second subrepresentation is a **trajectory frame:** motion along a path, with beginning and end states, as with prepositions, 'into the bucket', 'along the road'.
   3. A third subrepresentation is (state) **transition frames**, here, the increase/decrease of quantities (transition frames)

Let us first see how to write simple **transition frames** and then **trajectory frames.**

2. **Transition frames**. These indicate changes in some state variable over time. So, another way to look at them is as a table like the following, where a row is the name of a state variable, and the columns indicate clock ticks, t1, t2, t3, etc. The entries are one of 10 possible 'changes': change, not change; increase, not increase;

etc. Let's try this for the example "The Army grew larger before declining and then reached a constant size before disbanding". There are 3 'changes', so 4 columns in the table, and one variable 'value':

	Time1	Time2	Time3	Time4
Size army				

3. **Trajectory frames.**
Finally, we want to be able to map input sentences into output trajectory frames. To do this, we write an STT that admits possible English-like trajectory patterns, and then outputs a trajectory representation (instead of the db representation). Let's take a look at the partial trajectory STT in the Pset handout to see what it is doing. We want as output an event structure, e.g., (1) the main verb followed by (2) the object, then (3) the set of paths. For example, `(go ball (path (to place table))`. Note that a path couldbe a *set* of paths: e.g., a path could be `(path (from (place on table)) (path (to (place in basket)))`. NOTE that this form for trajectories is <u>defined</u> for you – you must follow it by stipulation. The job of your STT is to <u>output</u> this form. The main place this is done is in the line labeled 'trajectory' below, where, after the %end statement, you are to fill in the 'template' that in fact spits out this form. So, it should be clear what three expressions, and in which order, the three blank spots should be filled in with: those forms that will

```
(define trajectory-trees
  '((?sentence ((?trajectory %end ?trajectory))
               ((?path %end ?path))
               ((?thing %end ?thing)))
    (?trajectory ?thing ?goverb ?path %end  `(,_______ ,________ ,_______))
    (?path ((?path-element ?path %end   )
          (?path-element %end 'fill-me-in)
          (%end 'fill-me-in)))
    (?path-element (?path-preposition ?place %end  `(,_____________  _____________)))
    (?place ((?thing %end 'fill-me-in)
           (?place-preposition ?place %end 'fill-me-in)))
    (?path-preposition ((out of %end 'outof)
                   (from %end 'from)
                   (via %end 'via)
                   (through %end 'through)
                   (toward %end 'toward)
                   (to %end 'to)
                   (into %end 'into)
                   (off of %end 'offof)
                   (across %end 'across)))
    (?place-preposition ((above %end 'above)
                    (below %end 'below)
                    (at %end 'at)
                    (by %end 'by)
                    (on %end 'on)
                    (in %end 'in)
                    (under %end 'under)
                    (over %end 'over)
                    (near %end 'near)))
    (?thing ((?determiner ?thing %end ?thing)
           (troops %end 'troops)
           (enemy %end 'enemy)
           (baghdad %end 'baghdad)
           (mosul %end 'mosul)
           (tikrit %end 'tikrit)
           (offensive %end 'offensive)
           (airsupport %end 'airsupport)
           (bullets %end 'bullets)))
```

```
    (?determiner ((a %end #f) (an %end #f) (the %end #f)))
    (?goverb ((go %end 'go)
              (advance %end 'advance)
              (retreat %end 'retreat)))))
```