Lecture Notes - Conceptual Dependency and Natural Language Processing
CS405

Misc Administrative Topics:

Just a reminder to get going on your projects! At this point you should be quite close to having a program that can play Billabong using minimax. The full move generator should certainly be implemented. The next step should be integrating your program with the game manager and tuning your heuristics.

Strong Slot-and-Filler Structures : Conceptual Dependency  (Chapter 6)

Conceptual Dependency (CD) theory was invented by Roger Schank at Yale University in the 1970’s as a cognitive, psychological approach to AI. This is really necessary in many interesting domains where people are involved (e.g., user modeling, processing of news stories, etc.)

The underlying mechanism that Schank used for representation is called Conceptual Dependency. The purpose of CD is a separate language for representing meaning without ambiguity. This facilitates drawing inferences. If we were writing a language translator (something people do well) then we’d have something like:

```
<table>
<thead>
<tr>
<th>French</th>
<th>German</th>
<th>English</th>
<th>Swahili</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>parsing</td>
<td>Conceptual Dependency</td>
<td>generation</td>
</tr>
</tbody>
</table>
```

Here, CD represents an intermediate “language”, intended to be a deep conceptual representation that has no ambiguities, that can be used to translate to any other format. Note that for general language translators, this is much easier than creating direct mappings from one language to another. Psychological evidence suggests the existence of some form of CD in humans, as well, although language effects also come into play.

Some early systems that were built using CD theory:

SAM - Script Applier Mechanism. SAM could read simple script-based stories and make inferences. Variations have been used for reading newspaper stories (FRUMP). More on SAM later.

PAM - Plan Application Mechanism. Program capable of lower-level inference resolution that could apply to situations in which SAM failed.
Talespin - A story generation program, given goals and plans, the characters went through the actions to achieve their goals. Also addressed emotional, physical states.

Politics - Political analysis program adopting either a conservative or liberal viewpoint.

OpEd - Program capable of reading editorials and analyzing positions and argument structure.

All of these systems use the framework developed under Conceptual Dependency.

CD is based upon events and actions. Every event (if applicable) has:
    an ACTOR
    an ACTION performed by the Actor
    an OBJECT that the action performs on
    a DIRECTION in which that action is oriented

These are represented as slots and fillers. In English sentences, many of these attributes are left out. Consider the sentence:

1. “Kenrick went to New York.”

The object is missing here; we can fill it in with a rule that says the actor is the object with the word “go”. Thus, in CD we would have something like “Kenrick went Kenrick to New York”

2. “Mary fell.”

In this case, the actor is missing. Mary didn’t really do the falling, another force, gravity, has acted on Mary and pushed her in a direction, to the ground.

3. “John amazed Mary”

If we want to capture the actor, action, object, and direction this sentence gives part of it. It tells us John is the actor, and Mary is the recipient. However, it doesn’t specify what the action was. What the sentence does specify is that John did some unknown action that resulted in a state change within Mary. Note that one of the challenges in AI is to fill in unknown actions like this from previous sentences.

Schank and Abelson defined 11 primitive Actions they originally hoped would be sufficient to represent arbitrary sentences:

ATRANS: Abstract Transfer, e.g. give
PTRANS: Physical Transfer of location, e.g. go
PROPEL: Application of physical force to an object, e.g. push
MOVE: Movement of a body part by its owner, e.g. hit. Instrumental Act.
GRASP: Grasping of an object by an actor, e.g. clutch
INGEST: Ingesting an object by an animal, e.g. eat or drink
EXPEL: Expulsion of something by an animal, e.g. spit or cry or bleed
MTRANS: Transfer of mental information, e.g. tell
MBUILD: Creation of new information, e.g. mental construction or decision
SPEAK: Production of sounds, e.g. say. Instrumental Act.
ATTEND: Focusing sense organ toward stimulus, e.g. listen or look.
Instrumental act.
DO: Any action, used for unknown actions

Graphical structure of a CD event:

Can use these to link together sequences of events. Note that there are other modifiers, such as k for continuous events. The focus here is not on the details, but understanding how to make basic constructions of CD.

Let’s represent some examples, starting with the ones we have already done.

1. Kenrick went to New York.

2. Mary fell.
3. John amazed Mary.

More examples:

4. John saw Mary.


Here, CP is the “Cognitive Processor” of John, or John’s brain.
7. John shot up heroin.

This representation naturally leads to inferences. In fact, it is often difficult to keep inferences out of the representation process.

Consider the sentence:

8. John beat Mary with a bat.

The resulting condition that Mary was in a lower physical state is actually an inference. The sentence alone doesn’t say that Mary was hurt. However, it is something we have inferred. Normally we would leave this out of the CD representation, and use inference rules to figure out that Mary was hurt. For example, a rule could say that Phys.Contact with a person and a hard object results in a lower physical state.
9. Mary gave John a bat.

We can combine CD events as objects.

10. Mary told Kenrick that she gave John a bat.

11. Wile E. Coyote decided to kill the Road Runner.

There is a tendency to become ad hoc and make up our own definitions; this is ok as long as we are consistent and can still access the essential primitive actions.

Examples of sentences where CD is difficult to represent?
Scripts

Given a knowledge base in CD, or parsed English sentences into CD, what can we do with them? One application is to couple CD with the notion of scripts. A script is a stereotypical sequence of events in a particular context. These are sequences we have learned over time. They are similar to scripts for a play or movie, and contain actors, props, roles, scenes, and tracks.

Consider the stereotypical script of going to eat at McDonalds (do you remember the last time you ate fast food? How about the time before that?) There are set actors:

- cleaning guy
- cashier
- manager

There are set props:
- ketchup, mustard dispenser
- signs
- cash register
- tables
- drink machine

There is a stereotypical sequence of events:
- Wait in line
- Give order
- Pay money
- Receive food
- Sit down
- Eat
- Bus own table

Sometimes there are deviations to the script; e.g. going to the bathroom, or modifications to the script like getting a drink before receiving food. People use existing scripts or cases to learn new cases; eventually new cases may become new scripts.

Example: Let’s define a simple shopping script.

Actors: Shopper, Clerk
Objects: Merchandise
Location: Store
Sequences (in Lisp format):

(PTRANS (Actor ?Shopper) (Object ?Shopper) (To ?Store))
(PTRANS (Actor ?Shopper) (Object ?Bought) (To ?Shopper))
(ATRANS (Actor ?Store) (Object ?Bought) (From ?Store) (To ?Shopper))
(ATRANS (Actor ?Shopper) (Object (Money)) (From ?Shopper) (To ?Store))
(PTRANS (Actor ?Shopper) (Object ?Shopper) (From ?Store))

What do these mean in English?

How this helps us? Let’s say we now get an input story in the form of:

“Jack went to Target. Jack got a DVD player.”

We can represent this as:

(PTRANS (Actor (Jack)) (Object (Jack)) (To (Target))
(PTRANS (Actor (Jack)) (Object (DVD-Player)) (To (Jack))

These CD sequences match the script and would activate it. The rest of the script is inferred, so we can now answer questions like:

“Did Jack pay any money?”

By just looking up the CD form of the question:

(ATRANS (Actor (Jack)) (Object (Money))

this matches in the instantiated script, and our program can spit back “Yes!” and even give who Jack paid the money to (Target). In short, this allows us to do question/answering on unspecified events.

The script also helps us perform disambiguation. Once we know who the shopper is, we can use that to fill in the rest. Consider the input sequence:

“Jack went to Target. He got a DVD player.”

With the pronoun ‘he’, one way to disambiguation who the “he” refers to is through a script, which would match up the purchaser to the shopper.

In short, scripts allow:

- Inference of unspecified events, stereotypical sequences
- Disambiguation of actors and objects

Internally, scripts are often organized by their differences. For example, consider a general transaction. A more specific type of transaction could be a restaurant transaction.
More specific types are fast food vs. sit-down restaurants. If each of these events is stored as a node, they could be indexed based upon differences.

**Plans and Inference Rules**

In predicate calculus, inference rules were the major mechanism for making new useful conclusions. We can do the same thing in CD with a pair of CD statements to make a rule. These rules can be used to augment scripts, which will fail when we come across new situations that don’t match stereotypical sequences (e.g., what to do if a terrorist takes over the classroom. You probably don’t have a script for this, although you can certainly act and make decisions and predictions). This is more in line with traditional rule-based systems and inference.

In Schank’s experimental program PAM, the program could apply rules that activated low level events or entire scripts.

PAM created chains of inferences from various rules that would follow. When a chain was completed, the resulting CD was instantiated. For example, a rule could appear like:

- Grasping an object is a way to perform the plan of taking an object.
  
  (Instantiation
   (Take-Plan (Planner ?X) (Object ?Y))
   (Grasp (Actor ?X) (Object ?Y)))

- Taking something that is a book is with the goal of reading it.
  
  (Subgoal
   (Read-Plan (Planner ?X) (Object ?Y))
   (Take-Plan (Planner ?X) (Object ?Y)))

- Using a car is to achieve the goal of being at some location.
  
  (Subgoal
   (Use-Vehicle-Plan (Planner ?X) (Object ?Y))
   (Goal (Planner ?X) (Objective (Proximity (Actor ?X) (Location ?Y))))

- To use the restaurant script, first be at the location of the restaurant.
  
  (Subgoal
   (Use-Restaurant-$ (Planner ?X) (Restaurant ?Y))
   (Goal (Planner ?X) (Objective (Proximity (Actor ?X) (Location ?Y))))
If we had a small input story such as:

John picked up the Restaurant Guide. He drove to Humpy’s.

If we had the appropriate rules, we could infer a chain such as:

Picking up the restaurant guide → Take-Plan → Use-Plan → …
Picking up the restaurant guide → Take-Plan → Possess-Plan → …
Picking up the restaurant guide → Take-Plan → Read-Plan → Find-Restaurant-Plan → Use-Restaurant-$ → Proximity Goal at Restaurant
Driving to Humpy’s → Proximity Goal at Restaurant (If we know that Humpy’s is a restaurant) → Match with Restaurant guide.

By linking these sentences together, the system can answer questions like “Why did John drive to Humpy’s?” (To be at the proximity of the restaurant to use the restaurant plan). Additionally, we can use the rules to provide disambiguation of variables as with scripts.

In short, plans allow:
- Inference rules to connect CD events
- Disambiguation through variable instantiations

**Parsing Into CD : Conceptual Parsing**

So far, we have ignored the problem of parsing input text into CD. We’ve been assuming that we are already working in the CD domain. However, a more general system will have to parse input English text into the CD format.

One parsing technique is to assign a “packet” to each word with all of the sense definitions it may have. The packet watches for other words or context that came before or after it, and uses this context to determine the correct meaning of the word.

Example:

(Def-Word Jack
  (Assign *cd-form* (Person (Name (Jack)))
  *part-of-speech* Noun-Phrase))

Jack just takes the CD format of Person named Jack.

(Def-Word Lobster
  (Assign *cd-form* (Lobster)
  *part-of-speech* Noun
  *type* (Food)))
Definition for lobster is just a noun; we can include semantic information as well. Ideally this information (e.g. food, lobster) would also be indexed into a semantic hierarchy so that we have a better idea of what food and lobsters are.

```
(Def-Word Hair
  (Assign *cd-form* (hair)
    *part-of-speech* Noun
    *type* inanimate))
```

Just another sample definition, this time for Hair.

```
(Def-Word Had
  (Assign *part-of-speech* Verb
    *subject* *cd-form*
    (Next-Packet
      (Test (And
        (Equal *part-of-speech* Noun)
        (Equal *type* Food))
        (Assign *cd-form* (INGEST (ACTOR *subject*)
          (OBJECT *cd-form*)))))
      (Test (And
        (Equal *part-of-speech* Noun)
        (Equal *type* Inanimate))
        (Assign *cd-form* (POSS (ACTOR *part-of-speech*)
          (OBJECT *cd-form*)))))
)
```

This example disambiguates two meanings of the word “Had” by looking ahead at the next word or packet. If the next word is a type of food, then the INGEST CD is activated, otherwise if the next word is a type of inanimate object, then the “Possess” CD form is activated. Note that this is an example of forward disambiguation; we should also perform backward disambiguation by looking at the previous context of what has been seen so far in the sentence.

Now, if we have the entire sentence: “Jack had lobster”

When parsing this sentence from left to right, we retrieve the packet definition for each word. Then we assign any variables applicable, and put the next-packet on the stack for examination of future packets. We could also look backwards and see if previous packets have been examined for disambiguation purposes.

```
Jack: *pos* = Noun-Phrase
  *cd-form* = (Person (Name (Jack)))

Had: *pos* = Verb
  *subject* = (Person (Name (Jack)))
  Wait to see if next packet is a noun or food/inanimate before proceeding with which definition of Had.
```
Lobster: 
*pos* = Noun
*cd-form* = (lobster)
*type* = Food
This activates the first definition of HAD, so the definition is activated: (INGEST (Actor (Person (Name (Jack))))
(Object (Lobster)))

The main disadvantage of this approach is the complexity of the definitions; some words have many definitions, all of which must be carefully entered by the programmer. Additionally, there are many possible objects and other parts of speech that determine how a word should be disambiguated, making the process extremely difficult for large domains or general English. Note that people often don’t do this – they will fall for garden path sentences! But people are able to intelligently backtrack and choose another meaning if necessary (subliminally, there is evidence that more than one meaning is “activated”).

**Natural Language Processing**  (Chapter 13)

Typically, the process of parsing and understanding languages can be broken up into a number of different levels:

1. **Prosody.** This deals with the rhythm and intonation of the language.
2. **Phonology.** This examines the sounds that are combined to form language.
3. **Morphological Analysis.** This is the step of analyzing what is a word, what is punctuation, word tense, suffixes, prefixes, apostrophe, etc.
4. **Syntactic Analysis.** Essentially determine the part of speech of the words to see if it is valid. For example, the following sentence could be rejected from a syntactic analysis: “Tasty the Coon the fast slow green and the yes”
5. **Semantic Analysis.** Determining the meaning of the words to see if they make sense. A famous example is “Colorless green ideas sleep furiously” by Chomsky. This sentence is syntactically correct, but semantically meaningless.
6. **Pragmatic Analysis.** Reinterpret events to what they really mean. “Can I have a coke” at a restaurant is a request, not a yes/no question.
7. **Discourse Analysis.** A sentence may make sense individually, but not in the larger context. This phase examines the context of a particular sentence to see if it makes sense.

An example of the process is depicted in the diagram below. In this diagram we’re using CD as the semantic interpretation, but could use a semantic network, predicate calculus, etc.
The best system can ideally go across these boundaries in the process of parsing; for example, performing a semantic analysis can help while doing syntactic or even morphological processing. People also operate this way; we don’t wait for something to finish parsing before working on semantic analysis. We can see this by examining the mistakes that people make in reading “garden path” sentences like:

- The old man the boats.
- The horse raced past the barn fell.
- The player kicked the ball kicked him.

However, for ease of computing, usually computer programs separate parsing as these distinct phases.

Most of the work has been done in Syntactic Parsing. We can use many of the ideas used in compilers for parsing computer programs. A common technique is to define a grammar, and use that grammar to parse the sentences.
Here is a sample grammar for a subset of English:

Context Free Grammar:
S → NP VP
S → VP
NP → Determiner NP2
NP → NP2
NP → NP PP
NP2 → Noun
NP2 → Adj NP2
PP → Prep NP
VP → V
VP → V NP
VP → VP PP

Dictionary:
an : Determiner
arrow: Noun
flies: Noun, Verb
like : Preposition, Verb
time: Adj, Noun, Verb

Let’s build a Recursive Transition Network (RTN) from the grammar:
Solid circles indicate final states. Jumps can be taken directly without input. Ensure that there is only one link per label, e.g. a Deterministic Finite Automaton as opposed to a non-deterministic finite automaton.

Consider the sentence: “Time flies like an arrow”
Time flies like an arrow.
Three separate parses. To generate these parses from the grammar or RTN, we can do in a variety of methods, either bottom-up or top-down. Bottom-up parsing starts with the words and tries to go upwards in the tree, to see how they fit into the sentence. Top-down parsing starts with the sentence and tries to fill in values to complete them. Both processes are compute intensive since we have to keep track of all possible forks in the parsing process and examine them for viability (can combine bottom-up with top down filtering).

Combining Syntax and Semantic Knowledge

A common approach to construct a semantic representation from a syntactic parse is to recursively traverse the syntactic parse tree and construct a semantic parse tree. This is similar to what you saw in CS331 in constructing a semantic representation for a programming language from a syntactic parse, but in our case we'll use our knowledge base of frames to construct the semantic representation. For example, given the sentence "The dog bites the man"

We might parse this as follows:
Now let's define some semantic knowledge. The following is a type hierarchy used for this example:

![Type Hierarchy Diagram]

While we would define case frames for each concept, here are frames for the actions of like and bite. Notice the similarity to the conceptual dependency parser if we use actor instead of agent:

![Case Frames Diagram]

Now we are ready to define some pseudocode for semantic parsing. This code is dependent upon the grammatical rules available:

```plaintext
Process_Sentence
    Noun_Concept <-> Noun_Phrase()
    VP_Concept <-> Verb_Phrase()
    Bind Noun_Concept to agent in VP_Concept
```
Noun_Phrase Procedure
N ← Representation of Noun
If indefinite article and number singular, noun concept is generic
If definite article and number singular, bind marker to noun concept
If number plural, indicate that noun concept is plural

Verb_Phrase Procedure
V ← Representation of Verb
If verb has an object
  Noun_Concept ← Noun_Phrase()
  Bind concept for Noun_Concept to object of V

Here is what we have as we recursively parse the parse tree for "The dog bites the man."

1. ()
4. N = dog, singular
10. S = (bite (agent dog) (object man) (instrument (teeth (part dog))))
2. NP = ()
3. N = dog, singular
5. V = ?
7. NP = ()
8. N = man, singular
9. V = (bite (agent ?A) (object man) (instrument (teeth (part ?A))))

When the recursive process has finished, we've created the semantic representation of (Bite (Agent Dog) (Object Man) (Instrument (Teeth (Part Dog)))). This itself could be construed as a semantic tree:
Semantic, Discourse Analysis

Once the syntactic processing is complete, we are left with a number of parse trees. Semantic analysis can help us disambiguate which tree is correct. In many cases, we can get useful things out of the multiple parse trees; humans are notorious for selecting just one parse. This is actually the basis for many types of humor, as well as garden path sentences (“I had Turkey Jones over for lunch”. After reading “I had Turkey” most people think of the food turkey, not a person whose first name is Turkey.)

Semantic and discourse analysis composes most of the things we discussed in CD. A way to use the meaning of the words to further disambiguate what is happening. Semantic analysis can rule out many interpretations, such as that of “Time flies” being a type of fly, where Time is an adjective.

Scripts are one method of discourse analysis; they use the previous context and previous sentences to interpret new sentences. All steps together are required for a complete understanding of input text. However, portions may be used alone to address many problems. Additionally, often domains can be simplified to a point where a grammar may be constructed for it and the appropriate understanding tasks can be applied.

A good example of a complete system is on the web at http://www.sls.lcs.mit.edu. The MIT speech group has created a system that can perform speech recognition. The recognizer recognizes individual words. These words are put together into a sentence. Parsing the sentence can disambiguate words that were recognized incorrectly; the words are parsed into distinct trees, and the most likely parse tree is picked via semantic analysis. Sample domains include the ATIS air reservation system, and a system to give directions within Boston and Cambridge.

A demo that you can actually try is the MIT Jupiter system. Jupiter knows about the weather around the world. You can ask Jupiter questions, such as "What is the weather like in Anchorage?" or "Where is it snowing now?" You can try it by calling 1-888-573-TALK. Note that if you call, your voice will be recorded and used for future speech recognition research.