NYU

Partial Parsing

CSCI-GA.2590 – Lecture 5A

Ralph Grishman

Road Map

- Goal: information extraction

- Paths
  - POS tags → partial parses → semantic grammar

  - POS tags → full parses → semantic interp. rules

Road Map

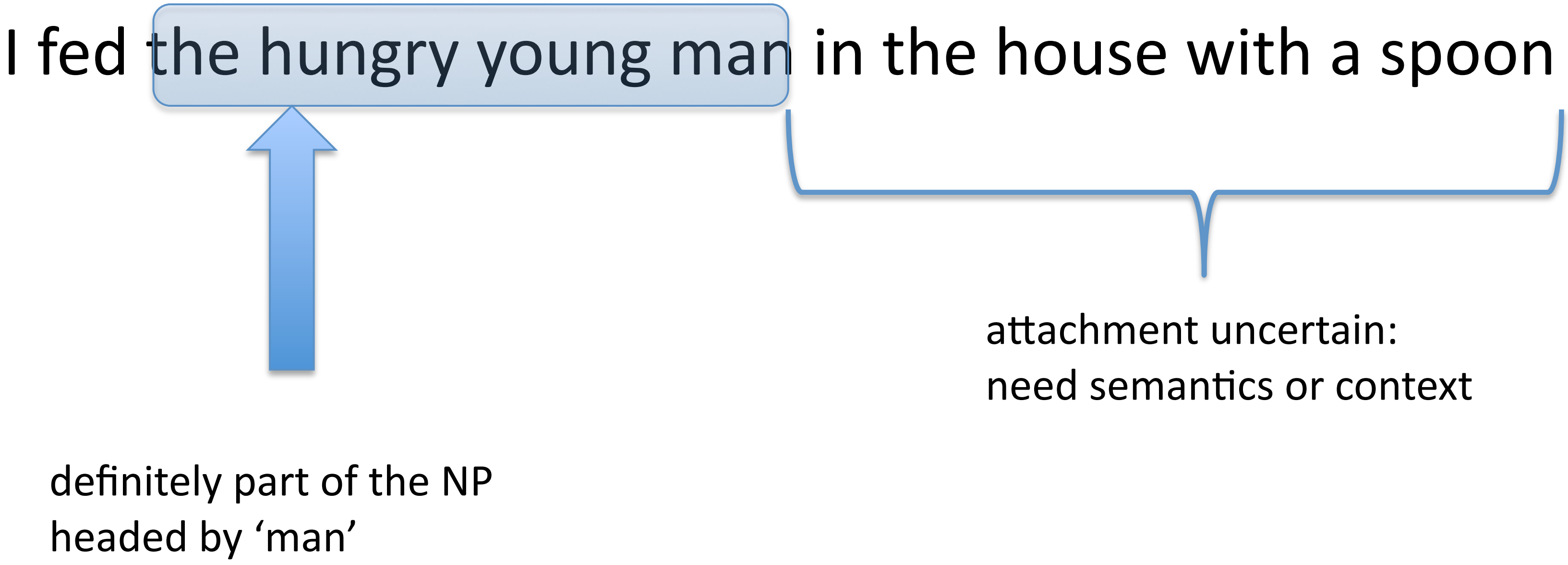
- Goal: information extraction

- Paths
  - POS tags → partial parses → semantic grammar
  - POS tags → full parses → semantic interp. rules

Partial Parses (Chunks)

- Strategy:
  - – identify as much local syntactic structure as we can simply and deterministically
    - general (not task specific)
    - result are termed chunks or partial parses
  - – build rest of structure using semantic patterns
    - task specific

Partial Parses (Chunks)



I fed the hungry young man in the house with a spoon

attachment uncertain:
need semantics or context

definitely part of the NP
headed by 'man'

Partial Parses (Chunks)

We will build and use two kinds of chunks:

- noun groups
  - head + left modifiers of an NP
- verb groups
  - head verb + auxiliaries and modals
    [“eats”, “will eat”, “can eat”, …]
    (+ embedded adverbs)

(we will use the terms ‘noun groups’ and ‘noun chunks’ interchangeably)

Chunk patterns

- Jet provides a regular expression language which can match specific words or parts of speech
- We will write our first chunker using these patterns

Chunk patterns: noun groups

```
ng := det-pos? [constit cat=adj]* [constit cat=n] |
        proper-noun |
        [constit cat=pro];

det-pos :=[constit cat=det] |
        [constit cat=det]? [constit cat=n number=singular] "'s";

proper-noun :=([token case=cap] | [undefinedCap])+;
```

Chunk patterns (verb groups)

```
vg :=[constit cat=tv] |
        [constit cat=w] vg-inf |
        tv-vbe vg-ving;
vg-inf :=[constit cat=v] |
       "be" vg-ving;
vg-ven :=[constit cat=ven] |
       "been" vg-ving;
vg-ving :=[constit cat=ving];
tv-vbe :="is" | "are" | "was" | "were";
```

Assembling the pipeline

tokenizer → POS-tagger → chunker

Tipster Architecture

Want a uniform data structure for passing information from one stage of the pipeline to the next

In the Tipster architecture, basic structure is the *document* with a set of *annotations*, each consisting of

- a type
- a span
- zero or more features

Each *annotator* (tokenizer, tagger, chunker) reads current annotations and adds one or more new types of annotations

Tipster Architecture

- Offset annotation means original document is never modified
  - benefit in displaying provenance of extracted information
- Document + annotations widely used
  - JET
  - GATE (gate.ac.uk – Univ. of Sheffield)
  - UIMA (uima.apache.org)
- but not universally: NLTK Python toolkit

Adding Annotations

My cat is sleeping

type	start	end	features

Adding Annotations: tokenizer

My cat is sleeping

type	start	end	features
token	0	3	case=forcedCap
token	3	7	
token	7	10	
token	10	19	

Adding Annotations: POS tagger

My cat is sleeping

type	start	end	features
token	0	3	case=forcedCap
token	3	7	
token	7	10	
token	10	19	
constit	0	3	cat=det
constit	3	7	cat=n number=singular
constit	7	10	cat=tv number=singular
constit	10	19	cat=ving

Adding Annotations: chunker

My cat is sleeping

type	start	end	features
token	0	3	case=forcedCap
token	3	7	
token	7	10	
token	10	19	
constit	0	3	cat=det
constit	3	7	cat=n number=singular
constit	7	10	cat=tv number=singular
constit	10	19	cat=ving
ng	0	7	
vg	7	19	

Jet pattern language (1)

- Matching an annotation:

  [type feature = value …]

  - must be able to *unify* features in pattern and annotation
  - can have nested features: feature = [f1 = v1   f2 = v2]
- Matching a string

  “word”
- Optionality and repetition

  X ?   (optionality)

  X * (zero or more X’s)

  X + (one or more X’s)

Jet pattern language (2)

- Binding a variable to a pattern element:

  [constit cat=n] : Head

- Adding an annotation

  when *pattern* add [ng];

- Writing output

  when *pattern* write "head =", Head;

Setting Up The Pipeline

```
# JET properties file to run chunk patterns
Jet.dataPath          = data
Tags.fileName         = pos_hmm.txt
Pattern.fileName1     = chunkPatterns.txt
Pattern.trace         = on


processSentence       = tagJet, pat(chunks)
```

pipeline stages
(tokenization implicit for interactive use)

Processing a Document

```
# JET properties file
#   apply chunkPatterns to article.txt
Jet.dataPath            = data
Tags.fileName           = pos_hmm.txt
Pattern.fileName1       = chunkPatterns.txt
Pattern.trace           = on


JetTest.fileName1       = article.txt
processSentence         = tokenize, tagJet, pat(chunks)
WriteSGML.type          = ngroup
```

split doc into sentences, then runs processSentence on each

Viewing Annotations

- can be activated through Jet menu:

tools : process documents and view …

- provides color-coded display of annotations

Corpus-Trained Chunkers

- We know two ways of building a sequence classifier which can assign a tag to each token in a sequence of tokens: HMMs and TBL
- Can we use a sequence classifier to do chunking? Assign N and O tags:

```
I gave a book to the new student.
N O    N N    O  N   N   N
```

- sequence of one or more consecutive Ns = a noun group

A Problem

- How about

```
I gave the new student a book
N O    N   N   N       N N
```

- 2 noun groups or 3?

BIO Tags

- A solution: 3 tags
  - B: first word of a noun group
  - I: second or subsequent word of a noun group
  - O: not part of a noun group

```
I gave the new student a book
B O    B   I   I       B I
```

- To tag noun and verb groups, need 5 tags: B-N, I-N, B-V, I-V, and O