Human Language Parsing

• We understand language incrementally, word-by-word

• How precisely do people construct interpretations?

• We must resolve local and global ambiguity

• How do people decide upon a particular parse/interpretation?

• Why are some grammatical sentences so difficult to comprehend

• Ambiguity: the human parser has been misled, takes time to recover

• Resources: the human parse has exceeded available memory
Human Language Parsing

• We understand language **incrementally**, word-by-word
  • How precisely do people construct interpretations?

• We must resolve local and global **ambiguity**
  • How do people decide upon a particular parse/interpretation?

• Why are some grammatical sentences so difficult to comprehend?
  • Ambiguity: the human parser has been misled, takes **time** to recover
  • Resources: the human parse has exceeded available **memory**

The Problem

• How do people recover the meaning of an utterance in real-time?

"The man held at the station was innocent"
Reading time studies

• We can use controlled experiments of reading times to investigate local ambiguity resolution

• (a) The man held at the station was innocent (LA)

• (b) The man who was held at the station was innocent (UA)

• Compare the reading times of (b) where there is no ambiguity, with (a) to see if and when the ambiguity causes reading difficulty.

• Need a “linking hypothesis” from theory to measures

• Can then manipulate other linguistic factors to determine their influence on on RTs in a controlled manner

Summary

• People construct an interpretations, word-by-word

• People must resolve ambiguity

• Sometimes we must revise our interpretation of the sentence so far

• Reading time measures can tell us about how/when this occurs

• Different phenomena are found in different measure

• We can design experiments which exploit these methods (and others!) to investigate the underlying processing architectures and mechanisms
Parsing Mechanisms

- Syntactic processing requires a solution to the problem of:
  - How structures are **incrementally** constructed
  - How local and global **ambiguity** is resolved

- Incremental Parsing
  - Top-down; Bottom-up; Mixed strategies

- Ambiguity and parsing:
  - Serial (deterministic/non-deterministic)
  - Parallel (bounded/unbounded)

Parsing Algorithms for PSGs

- Algorithms to recover the parse tree for an utterance vary ... 
  - left-to-right, head-driven, right to left
  - top-down, bottom-up, mixed
  - deterministic, serial, parallel

- Processing complexity:
  - Time: what time is required to parse a sentence as a function of sentence length, grammar size?
  - Space: how much memory does the parser require?
Bottom-up Parsing

- “The woman reads”

### Shift-reduce Algorithm

1. Initialise \( \text{Stack} = [] \)
2. loop: Either shift:
   - Determine category, \( C \), for next word in sentence;
   - Push \( C \) onto the stack;
3. Or reduce:
   - If categories on the \( \text{Stack} \) match the RHS of a rule:
     - Remove those categories from the \( \text{Stack} \);
     - Push the LHS category onto the \( \text{Stack} \);
4. No more words to process?
   - If \( \text{Stack} = [S] \), then done;
5. Goto loop
Simple example: Bottom-up

```
NP                  VP
  Det  N     VP
  the  man   read

Det       N       V             NP
g            g            tu

NP
  Det  N
  the  man  read
g

g
g
g
```

Top-down Parsing

“The woman reads”

```
S  [S]    S  [NP,VP]    S  [Det,N,VP]
  NP   VP  NP   VP   NP   VP
  Det  N   Det  N   Det  N  V
  the  man  the  woman  the  woman  reads
```
Top-down Algorithm

1. Initialise $Stack = [S]$
2. If $\text{top}(Stack)$ is a non-terminal, $N$:
   - Select rule $N \rightarrow RHS$;
   - pop($N$) off the stack and push($RHS$) on the stack;
3. If $\text{top}(Stack)$ is a pre-terminal, $P$:
   - Get next word, $W$, from the input;
   - If $P \rightarrow W$, then pop($P$) from the stack;
   - Else fail;
4. No more words to process?
   - If $Stack = []$, then done;
5. Goto 2

Simple example: Top-down

```
the man read every book
```

```
S → NP VP
NP → Det N
VP → V NP
```
Evaluating top-down & bottom-up

- Are these parsers psychologically plausible?
- Incrementality:
  - Bottom-up: no
  - Top-down: yes
- Input-driven:
  - Bottom-up: yes
  - Top-down: no + problems with left-recursion

A Psychologically Plausible Parser

- Left-Corner Parsing
- Rules are ‘activated’ by their ‘left-corner’

- Combines input-driven with top-down
- There is a ‘class’ of LC parsers
An example LC parse

“The woman read the book”

Is this incremental?

Simple example: Left-Corner
Evaluating the LC Parser

- Variations: **arc-standard** versus **arc-eager**

![Diagram of sentence structures]

- Affect on ambiguity resolution for arc-eager:
  - Commitment to attachments is early, before daughters are completely built.

Quick experiment

- “The mouse died”

- “The mouse that the cat chased died”

- “The mouse that the cat that the dog bit died”

- “The mouse that the cat that the dog bit chased died”
Incrementality and Memory

- It wasn't incrementality that led to the LC algorithm, but memory load:
  - “The mouse died”
  - “The mouse the cat chased died”
  - “The mouse the cat the dog bit chased died”

    (Or: “The mouse that the cat that the dog bit chased died”)

- Grammatical, not ambiguous, what's the problem?
- Memory load: too high for centre embedding
  - “[The mouse [the cat [the dog bit] chased] died]”

Memory Load in Parsing

- Left-embedding is easy:
  - [[[John’s brother]’s car door]’s handle] broke off.

- Right-embedding too:
  - John believes [Bill knows [Mary said [she likes cats]]]

- Centre-embedding is hard:
  - [The mouse [the cat [the dog bit] chased] died]

- Memory load for parsers:
  - Top-down: LE: hard   CE: hard   RE: easy
  - Bottom-up: LE: easy   CE: hard   RE: hard
  - Left-corner: LE: easy   CE: hard   RE: easy
Evaluating the LC Parser

- Variations: **Arc-standard** versus **Arc-eager**

![Diagram of a sentence structure with nodes labeled S, NP, VP, and V. The sentence is: "the man knew the dog ...". The arc-standard parser shows 3 arcs, while the arc-eager parser shows 1 arc.]

**Summary of Behaviour**

<table>
<thead>
<tr>
<th>Node</th>
<th>Arcs</th>
<th>Left</th>
<th>Centre</th>
<th>Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Top-down</td>
<td>Either</td>
<td>O(n)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>Shift-reduce</td>
<td>Either</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Left-corner</td>
<td>Standard</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Left-corner</td>
<td>Eager</td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
<tr>
<td>People</td>
<td></td>
<td>O(1)</td>
<td>O(n)</td>
<td>O(1)</td>
</tr>
</tbody>
</table>
Comments on Left-Corner

- Mixed data-driven and hypothesis driven approaches
  - Eager corresponds to composition of partial structures
- **Arc Standard**: less ambiguity
  - attach when constituents are complete: safer
  - delayed attachment means more is kept on the stack
- **Arc Eager**: less memory
  - early composition reduces stack growth
  - eager attachments are less bottom-up

Ambiguity in Parsing

- Rule selection: *what if more than one rule can be selected?*
  - Local ambiguity: a parse derivation may fail later
  - Global ambiguity: multiple parses can succeed
- How can we handle local and global ambiguities during parsing:
  - Backtracking
  - Parallelism
  - Determinism
  - Underspecification
Ambiguity in Parsing

- Rule selection: *what if more than one rule can be selected?*
  - Local ambiguity: a parse derivation may fail later
  - Global ambiguity: multiple parses can succeed

- How can we handle local and global ambiguities during parsing:
  - **Backtracking**
  - **Parallelism**
    - Determinism
    - Underspecification

Backtracking Parsers

- Parsing is a sequence of *rule selections*
- If at one point, more than one rule can be applied, this is called a *choice point*
  - Make a decision, based on some selection rule
  - If subsequently parsing ‘blocks’, return to a choice point and re-parse from there
- Which choice point to return to?
  - usually the last, why?
  - what other choice point selection rules could be used
Backtracking: an example

Bill reads

Parallel Parsers

- Build parse trees through successive rule selections

- If more than one rule may be applied, create a new parse derivation for each possibility

- Pursue all parses in parallel

- If any of the parses 'blocks', discard it

- Because of multiple local ambiguities, the number of parallel derivation grows exponentially

- Bounded parallelism: pursue a fixed number

- How do we choose which ones to keep?
Parallel: an example

Bill reads

Parse 1

Parse 2

Discard

Pursue