Advanced Natural Language Processing Lecture 10 Parsing

Mark Steedman

November 3, 2009

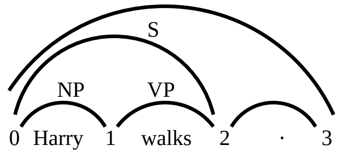
School of informatics

Tabular Parsing with Dynamic Programming

- Two notations:
  - Tables

Harry	NP	S
	walks	VP
		.

  - Charts



Bottom-Up Parsing

- The basic idea:

1. Initialise the chart to the empty chart, and make a pointer point to position 0 in the string, before the first word.
2. Add entries corresponding to all categories of the word which starts at the pointed-to position. Make the pointer point to the next position in the sentence word in the sentence.
3. As long as there is a pair of entries in the chart that can reduce, do the reduction and add an entry representing the result to the chart, unless an equivalent entry is already present.
4. Goto step 2

CKY Algorithm

- CKY originally defined for Chomsky Normal Form ($A \rightarrow B \quad C/A \rightarrow a$) only but generalized by Gray and Harrison (1972); Harrison (1978)
- When we shift a new lexical category for the jth word, we can only induce new reductions yielding categories whose right boundary is at j.
- An efficient way of carrying out step 3 is to ask for all i where $0 \leq i \leq (j-2)$ whether there are any such reductions.
- This in turn means asking for all k where $i \leq k \leq j$ whether there are entries spanning (i,k) and (k,j) that reduce.
- Since adding a new entry (i,j) during this process may itself enable further reductions, it is necessary to compute the new entries (i,j) bottom-up – that is, by starting with $i = j-2$ and stepping down to $i = 0$.

CKY Algorithm

- Hence we can state the algorithm more formally as follows:

```
1. for $j := 1$ to $n$ do
   begin
   $t(j,j) := \{A|A$ is a lexical category for $a_j\}$

2. for $i := j-1$ down to 0 do
   begin

3. for $k := i$ down to 0 do
   begin
```

```
$t(k,j) := \{A|$ for all $B \in t(k,i), C \in$ end
            $t(i+1,j)$
            such that $B\ C \Rightarrow A$ for
            some combinatory rule in
            $R$
            and not $already\text{-}$
            $present(A,k,j)\}$
end
end
```

CKY Algorithm

- Three nested loops implies $O(n^3)$ recognition and parsing.
- For sensible CF grammars, the subsumption check is redundant.
- When we generalize to trans-CF grammars it may not be.

Example

- Grammar:

$$\begin{array}{lcl} S & \longrightarrow & SS|AA|b \\ A & \longrightarrow & AS|AA|a \end{array}$$

- String: aabb

- Table:

	A	S,A	S,A	A,S
		A	A	A
			S	S
				S

Earley Algorithm

- A close relative of $LR(k)$.

- Grammar:
$$\begin{array}{lll} S & \longrightarrow & NP \quad VP \\ NP & \longrightarrow & Det \quad N \\ Det & \longrightarrow & NP \quad 's \\ NP & \longrightarrow & Gilbert \\ N & \longrightarrow & friend \\ \ldots. & & \end{array}$$

- The basic idea:
  Top-down depth-first recursive search of all expansions starting from the start symbol S.

- The Problem:

  Unlike bottom-up, naive top-down is blocked by infinite regress on recursive rules like those above.

- The solution:
  Limit recursion to things that there is actually some support for in the string.

Earley Algorithm

- A state of the algorithm can be represented by a set of *edges*, each representing:

  1. An (active or inactive) grammar rule (aka active or inactive edge)
  2. A position in the rule that we have found everything to the left of, represented as a dot.
  3. A k-symbol string of lookahead symbols
  4. A pointer to the string position marking the left end of the constituent defined by the grammar rule.

Earley

- Each Earley rule or edge in a state S_j with left hand end i is directly equivalent to a chart parsing edge spanning i, j. If the dot is on the right hand end, it is an inactive edge.

- For example, the start state S_0 for a parser with lookahead $k = 1$ includes the following edge:

$$Init \rightarrow .S \dashv\dashv 0$$

- This is the active arc from 0 to 0, seeking an S. ($\dashv$ is a special end marker, and we arrange that all sentences end in $k+1$ of them.)

Earley

- We generate new states from old by three operators:

  1. *Predict:* For current state S_i and an active edge A, if there is a non terminal N to the right of the dot, we add to S_i an active edge $N \rightarrow .\alpha\kappa\pi$ for each rule in the grammar having N on the LHS and α on the RHS. κ is a string of k terminals that the grammar allows to follow N according to the rule A, and π is the left-hand end position of the original rule, *unless an identical active edge is already present*.
  2. *Scan:* For current state S_i and an active edge A, if there is terminal T to the right of the dot, then T is compared with the next item X_{i+1} in the string. If it is the same, then an edge like A with the dot advanced to the right of T is added to S_{i+1}.

3. *Complete:* For current state S_i and an active edge A with left end pointer π, if there is nothing to the right of the dot for some state whose rule has non-terminal N on its LHS, compare the lookahead string κ with the next k symbols. If they match take every state in the pointed to state S_π which has the dot to the left of N, and add a corresponding state to the current set S_i with the dot to the right of N.

Earley

- NOTE 1: If every non terminal in the grammar is followed by a terminal, and $k = 1$, as happens to be the case in this example, then the κ induced by the *Predict* step will be that terminal. However, in general the possible κs will have to be precomputed beforehand. This point may not immediately be clear from Earley's paper.

- NOTE 2: The step *Complete* has the effect of allowing a basically breadth first top-down analyser to share like subparses. That is, when we have found an NP, we move the dot on in all active rules that are currently looking for an NP.

- NOTE 3: The step *Complete* does not remove any edges from previous states.

Earley

- Consider parsing the sentence *Gilbert walks*, with lookahead $k = 0$:

- *Predict* will ensure that among other rules, the following active edges are represented in state S_0:

$$\begin{array}{llllll}
Init & \rightarrow & . & S & 0 & \\
S & \rightarrow & . & NP & VP & 0 \\
NP & \rightarrow & . & Det & N & 0 \\
Det & \rightarrow & . & NP & 's & 0 \\
NP & \rightarrow & . & gilbert & 0 & \\
\ldots. & & & & &
\end{array}$$

  The clause in italics at the end of *predict* stopped it adding an infinite number of edges like #4, because of the left recursive rule.

- *Scan* operates on edge #5, adding the following edge to the next state S_1: $NP \quad \rightarrow \quad gilbert \quad . \quad 0$ Since no further rule applies to S_0, we start work on this set.

Earley

- *Complete* applies to the above edge to add active edges corresponding to all edges in S_0 that were looking for an NP, so S_1 looks like this:

  $NP \;\rightarrow\; gilbert \;.\; 0$

  $S \;\rightarrow\; NP \;.\; VP \;\; 0$

  $Det \;\rightarrow\; NP \;.\; 's \;\; 0$

  $\ldots.$

- Note that we have an active edge from 0 to 1 looking for a VP, which will give rise to further predictions. If the next word is *walks*, this active edge will find its VP, and ultimately result in state S_2's including the following edge:

  $\ldots.$

  $Init \;\rightarrow\; S \;.\; 0$

  $\ldots.$

Earley

- However, S_1 also includes an active edge from 0 to 1 looking for the terminal *'s*. If the sentence begins *Gilbert's friend . . .* , then *Scan* will add the following edge to S_2: $Det \rightarrow NP \ 's \ . \ 0$

- *Complete* will then add an active edge from 0 to 2 seeking a noun
  $Det \rightarrow NP \ 's \ . \ 0$
  $NP \rightarrow Det \ . \ N \ 0$

- When the noun is found, a similar process will lead to state S_3 including the following active edge parallel to the one in S_1, apart form spanning 0 to 3: $S \rightarrow NP \ . \ VP \ 0$

- If the sentence is *Gilbert's friend walks*, then the analysis wil complete in S_4, much as before.

- NOTE 4: The algorithm does not block on recursive rules.

Human and Computational NLP

- No handwritten grammar ever has the coverage that is needed to read the daily newspaper.

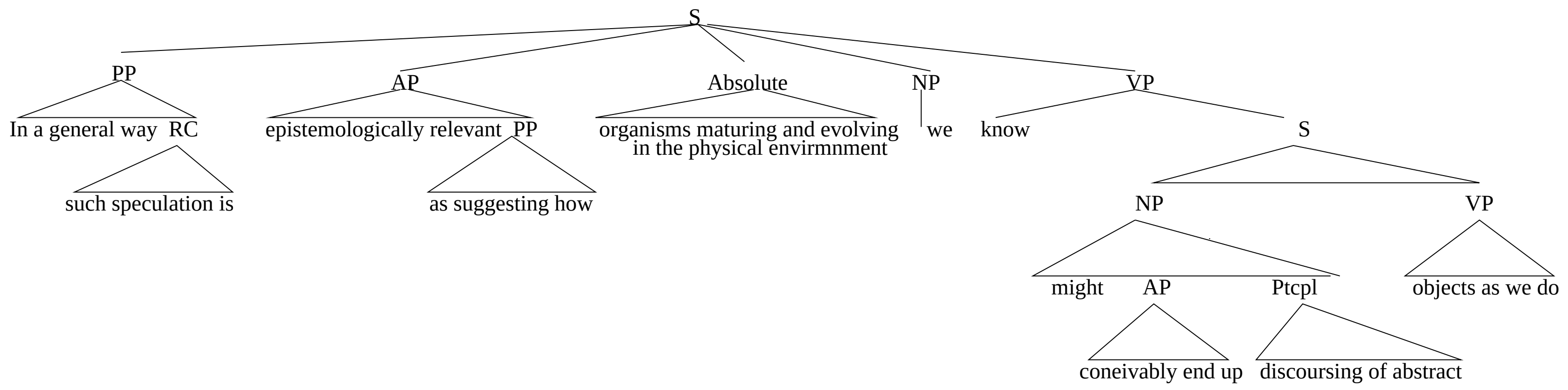
- Language is syntactically highly ambiguous and it is hard to pick the best parse. Quite ordinary sentences of the kind you read every day routinely turn out to have hundreds and on occasion thousands of parses, albeit mostly semantically wildly implausible ones.

- High ambiguity and long sentences break exhaustive parsers.

For Example:

- "In a general way such speculation is epistemologically relevant, as suggesting how organisms maturing and evolving in the physical environment we know might conceivably end up discoursing of abstract objects as we do." (Quine 1960:123).

- —yields the following (from Abney 1996), among many other horrors:



The Anatomy of a Parser

- Every parser can be identified by three elements:
  - A Grammar (Regular, Context Free, Linear Indexed, etc.) and an associated automaton (Finite state, Push-Down, Embedded Push-Down, etc.);
  - A search Algorithm characterized as left-to-right (etc.), bottom-up (etc.), and the associated working memories (etc.);
  - An Oracle, to resolve ambiguity.

- The oracle can be used in two ways, either to actively limit the search space, or in the case of an "all paths" parser, to rank the results.

- In wide coverage parsing, we usually have to use it in the former way.

Competence and Performance

- Chomsky (1957, *passim*), has always insisted on the methodological independence of "Competence" (the grammar that linguists study) and "Performance" (the mechanisms of language use).

- This makes sense: there are many more performance mechanisms than there are grammatical levels, and for any sentence there are many ways of uttering it.

- Nevertheless, Competence and Performance must have evolved as a single package, for what is a parser without a grammar, or a grammar without a parser/generator?

Human Sentence Processing

- "Garden path" sentences are sentences which are grammmatrical, but for which naive subjects fail to parse.

- Example (1a) is a garden path sentence, because the ambiguous word "sent" is analysed as a tensed verb:

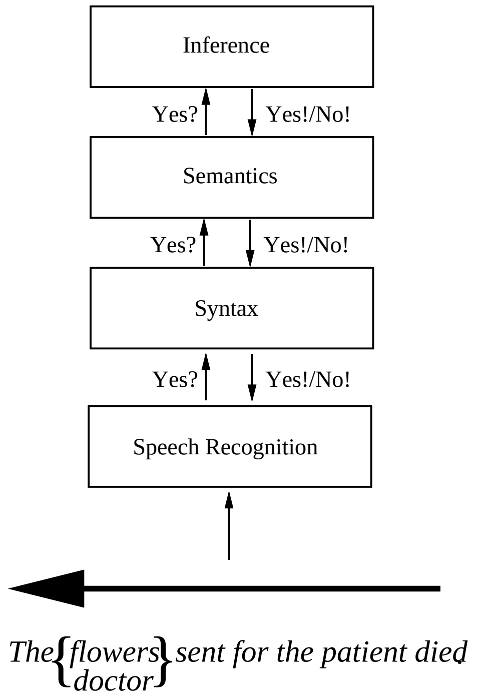
  (1) a. The doctor sent for the patient died.
  b. The flowers sent for the patient died.

- However (1b) is not a garden path.

- So garden path effects are sensitive to semantic content and pragmatic knowledge, (Bever 1970) and even to context (Altmann and Steedman 1988).

The Architecture of the Human NLP

- This requires a "cascade" architecture:



Human Sentence Processing

- This architecture embodies Incremental Fine-grain Parallel, “Weakly Interactive” Parsing

- The “Weak” interaction with semantics is where syntax proposes interpretations, and semantics and pragmatics and inference in context then rank them for plausibility (Crain, Altmann, et al.).

Human Sentence Processing

- **Contexts:** A burglar broke into a bank carrying some dynamite. He planned to blow open a safe . . .
  - **NP-attachment-supporting continuation**: . . . Once inside he saw that there was a safe with a new lock and a safe with an old lock.
  - **VP-attachment-supporting continuation**: . . . Once inside he saw that there was a safe with a new lock and a strongbox with an old lock.

Human Sentence Processing

- **Target Sentences:**
  - **NP-attached target**:
    The burglar blew open the safe with the new lock and made off with the loot.
  - **VP-attached target**:
    The burglar blew open the safe with the dynamite and made off with the loot.

Weakly Interactive Parsing (contd.)

- If garden paths are under the control of context, why do the classic garden path sentences garden path in the *neutral* or "null" context?
  - Because the null context *isn't* neutral. It is instead the *simplest* context compatible with the sentence being processed.
  - It is simpler to accomodate one safe than more than one, one horse rather than several horses, etc.
  - In the case of examples like *The horse raced past the barn fell* there are even more presuppositions to accomodate—like their being an activity which made one of the horses race alonfg a particular path.

Weakly Interactive Parsing (contd.)

- Can we afford to implement weak semantically interactive parsing in practice?
  - Only if we can model the knowledge of the domain completely.
  - Therefore, not for tasks like parsing the daily newspaper or arbitrarily-chosen web-pages.

Weak Interaction and Competence Grammar

- It is interesting that CCG's unorthodox approach to syntactic constituency means that most left prefix substrings of sentences are typable constituents, complete with an interpretation.

- For example, the fact that (2a,b) involve the nonstandard constituent [The doctor sent for]$_{S/NP}$, means that constituent is also available for the canonical sentence (2c)

(2) a. The patient that [the doctor sent for]$_{S/NP}$ died.
b. [The doctor sent for]$_{S/NP}$ and [The nurse undressed]$_{S/NP}$ the patient who had complained of a pain.
c. [The doctor sent for]$_{S/NP}$ the patient.

The Strict Competence Hypothesis

- This means that the spurious constitutent [#The flowers sent for]$_{S/NP}$ is also available with an interpretation, so that its semantic anomaly can be detected via the weak or filtering interaction, and the garden path in (1b) avoided, even under the following very strong assumption about the parser:
- The Strict Competence Hypothesis: the parser only builds structures that are licensed by the Competence Grammar as typable *constituents*.
- This is an attractive hypothesis, because it allows Competence Grammar and Performance Parser/Generator to evolve as a package deal, with parsing completely transparent to grammar.
- But is such a simple parser possible? We need to look at some real-life parsing programs.

References

Abney, Steven, 1996. "Statistical Methods and Linguistics." In Judith Klavans and Philip Resnik (eds.), *The Balancing Act*, Cambridge MA: MIT Press. 1–26.

Altmann, Gerry and Steedman, Mark, 1988. "Interaction with Context During Human Sentence Processing." *Cognition* 30:191–238.

Bever, Thomas, 1970. "The Cognitive Basis for Linguistic Structures." In John Hayes (ed.), *Cognition and the Development of Language*, New York: Wiley. 279–362.

Chomsky, Noam, 1957. *Syntactic Structures*. The Hague: Mouton.

Gray, J. and Harrison, Michael A., 1972. "On the Covering and Reduction Problems for Context-Free Phrase Structure Grammars." *Journal of the Association for Computing Machinery* 19:385–395.

Harrison, Michael, 1978. *Introduction to Formal Language Theory*. Reading MA: Addison-Wesley.

Quine, Willard van Ormond, 1960. *Word and Object*. Cambridge MA: MIT Press.