Ling/CSE 472: Introduction to Computational Linguistics

5/18/15
Parsing, unification, parsing with unification

Overview

- Earley parser
- Unification
- Earley with unification

Parsing

- Given a CFG and a sentence, whether the CFG accepts it, with what and how many structures, is a mathematical fact

- Given a CFG and a sentence, determining whether the CFG accepts it, with what and how many structures, is a *search problem*

- Parsing algorithms can be:

  - Top-down or bottom-up

  - Breadth-first or depth-first

  - “Best”-first or exhaustive

The Earley Parser

- For a sentence of N words, the chart contains N+1 cells.
- Each cell contains a list of states.
- A state consists of: a local subtree (‘edge’), information about the degree of completion of that subtree, and information about how much of the string corresponds to the subtree.
- For example (in dotted-rule notation):
  - S → •VP, [0, 0]
  - NP → Det • Nominal, [1, 2]
  - VP → V NP•, [0, 3]

The Earley Chart Parser, Outline

- Add the initial S (gamma → •S, [0,0]) to the chart at position 0.
- Loop, for each of the rest of the cells in the chart:
    - If the state is incomplete, and the category to the right of the dot is not a POS, add (if not redundant) new states to the chart in the current position for each rule that expands that category. These new states all have the dot at the beginning of the rule, and a span that starts and ends at the current position. (PREDICTOR)

The Earley Chart Parser, Outline

- Loop (continued):
    - If the state is incomplete, and the category to the right of the dot (‘B’) is a POS, and B is a possible POS for the next word in the string, add the rule B → word • (if not redundant) to the next cell in the chart. (SCANNER)
    - If the state is complete (dot all the way to the right), look in the cell corresponding to the beginning of the current state’s span for states which are currently seeking a daughter of the same category as the mother of this state. For each one of those, add a state (if not redundant) to the current cell, with the dot moved over one, and the span increased to the end of the current word. (COMPLETER)

Example: Kim adores snow in Oslo

- Toy grammar

$$S \rightarrow \text{NP VP}$$
$$\text{VP} \rightarrow \text{V NP}$$
$$\text{VP} \rightarrow \text{V}$$
$$\text{VP} \rightarrow \text{VP PP}$$
$$\text{PP} \rightarrow \text{P NP}$$
$$\text{NP} \rightarrow \text{NOM PP}$$
$$\text{NP} \rightarrow \text{NOM}$$

$$\text{NOM} \rightarrow \text{Kim}$$
$$\text{NOM} \rightarrow \text{snow}$$
$$\text{NOM} \rightarrow \text{Oslo}$$
$$\text{V} \rightarrow \text{adores}$$
$$\text{V} \rightarrow \text{snores}$$
$$\text{P} \rightarrow \text{in}$$

The Earley Chart Parser

- In which cell of the chart does one find the spanning edge(s)?
- Is the Earley algorithm top-down or bottom-up?
- Best-first or exhaustive?
- Breadth-first or depth-first?
- How does it handle ambiguity?
- How does it avoid inefficient re-parsing of subtrees?
- How would this alogrithm return the trees?

Unification

- Input: Two feature structures
- Output: Failure signal or the unique most general feature structure containing all info from both inputs

Unification algorithm

- Use graphs to represent feature structures
- Augment these structures with another layer, adding features ‘pointer’ and ‘content’
- Use pointers to merge the graphs representing the two input feature structures (why?)
- Unification is recursive (why?)
- Unification is destructive (why?)

Unification example

$$\begin{bmatrix} \mathrm{A} & \boxed{1}\begin{bmatrix} \mathrm{B} & \mathrm{c} \end{bmatrix} \\ \mathrm{D} & \begin{bmatrix} \mathrm{E} & \boxed{1} \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \mathrm{D} & \begin{bmatrix} \mathrm{E} & \begin{bmatrix} \mathrm{F} & \mathrm{g} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

- Represent each feature structure as a DAG
- Add the ‘pointer’ and ‘contents’ features
- Step through the unification algorithm to produce the result
- How would we have to alter this to handle typed feature structures?

Parsing with unification

- Associate feature structure constraints with rules
- Associate feature structures with edges
- Could just check after CFG parsing is done, but this is inefficient (why?)
- Instead: invoke unification when combining edges (COMPLETER)
- When deciding whether an edge to add is redundant, test is now *subsumption* (rather than identity): don't add edges that are *subsumed* by something already in the chart

Evaluation slide

- When we are evaluating parsing algorithms (rather than parsers incorporating specific grammars), what are we evaluating them for?
- What kinds of tests could we perform?
- What inputs do we need for the tests?

Reading questions

- I don't fully understand the DAG representations of unification. It seems like going through the structure would give you just one output (e.g. sg or 3rd), even when the input has two features (e.g. [NUMBER sg, PERSON 3rd]). Are you supposed to go through the DAG multiple times? If so, how do you know when to stop?

- In Unification, is there a way to check for compatibility without merging the two if it has not failed? Is it the case that we always want the merger if they are compatible?

- When using the unification algorithm with two identical arguments, why is it necessary to set the pointer of the first to the second? While it makes sense that you'd need to make one thing point to another to combine the two features, it's not clear from the text whether setting the pointer of the second to the first would also work.

Reading questions

- When and why should we use the unification operation of verbs? What will happen if the AGREE of two verbs can be combined but the VFORM cannot?
- How does conditional features work with unification? For example "If I were...", the word "I" is always singular, but the word "were" is mostly known to be plural. How do we define conditions in feature structures and does unification recognize it?

Reading questions

- Is it possible to construct a single type hierarchy tree that includes all possible feature structure constraints for a given language, and is this ever done? Or are the type hierarchies only made for individual categories of simple or complex types.

- Types are organized into a hierarchies, where more specific types inherit properties from more abstract ones. Are there any languages where it is difficult to organize these hierarchies?

Reading questions

- On page 524, they mention the lexical rule and how it can be used to capture generalizations and replace inheritance. Could you explain more about the lexical rule and how it's different from inheritance?

- What's an example of where a priority union operation would be used?

Reading questions

- Towards the end of the chapter, the book mentioned that recent work in unification grammar focuses on disambiguation. What are some mechanism that are used in unification grammars that focuses on the disambiguation problem? Can you give an example?

Overview

- Earley parser
- Unification
- Earley with unification
- Next time: probabilistic parsing