Lecture 8

Earley Parsing

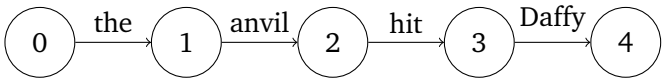
The CKY parser is a non-directional bottom-up parser with a chart, but chart parsing is in no way restricted to these parameters. In this chapter, we look at a directional chart parser with a top-down component: the Earley parser. The Earley parser combines a variety of techniques and insights we have encountered up to this point, and we will emphasize these connections by carefully working out the relation between Earley parsing and LC parsing. The Earley parser is also one of the fastest known algorithms for arbitrary CFGs, and in contrast to the CKY parser it does not require grammars to be in Chomsky Normal Form. In addition, Earley parsing has shown promise in some psycholinguistic modeling experiments (**?**).

1 Intuition

There are two intuitive perspectives of Earley parsing. We may either view it as a top-down parser where predictions are restricted by bottom-up reductions, or as a bottom-up parser where reductions are restricted by top-down predictions. In either case we are clearly dealing with a model that mixes top-down and bottom-up aspects, similar to the LC parser.

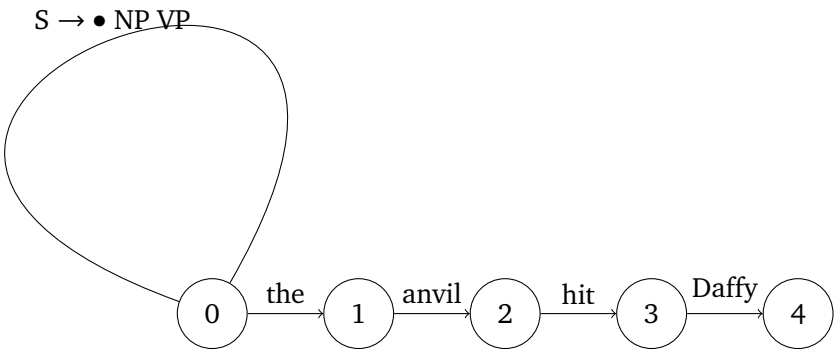
The basic idea is that the parser may freely make predictions like a top-down parser, while also using bottom-up information to quickly discard incompatible predictions before they have been fully explored. The parser moves through the input from left-to-right while trying to construct an arc that spans from the first position to the last. Arcs are just convenient abstractions of subtrees in this context. If the end position of one arcs is the start position of another arc, they can be combined into a larger arc, just like two subtrees can be combined into a single tree by attaching them below a new node. The strength of the Earley parser is how it avoids spending time on arcs that are bound to fail.

All of this is best explained through a concrete example. Suppose that we have our usual toy grammar and want to parse the input sentence *The anvil hit Daffy*, represented here as an indexed string.

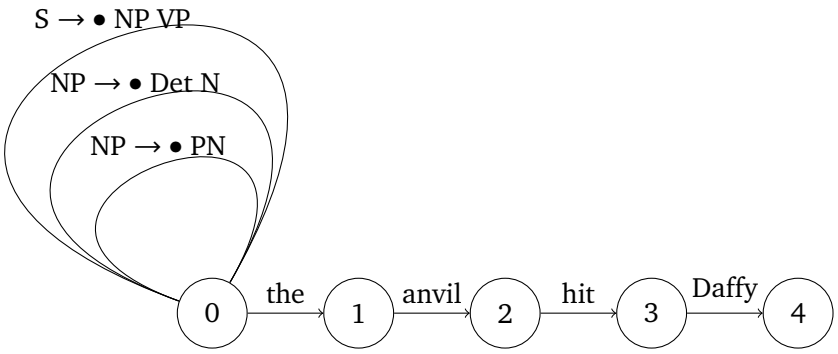


The parser will start out by conjecturing an arc from 0 to 0 that is labeled S → •NP VP. This is tantamount to the prediction that there will be an S-arc starting in position 0,

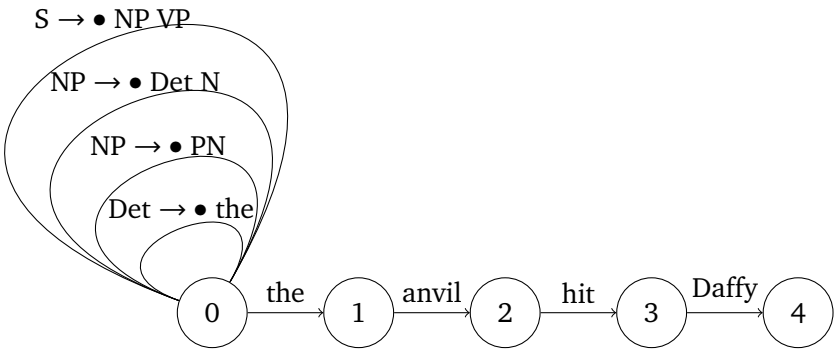
but so far we have not recognized an NP or a VP, so the confirmed part of the arc ends in position 0.



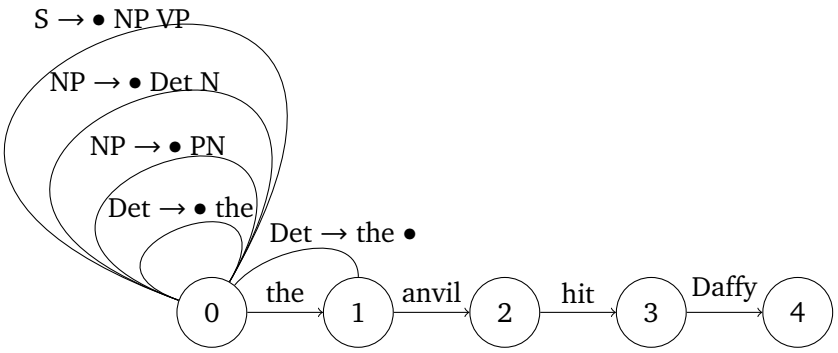
Since the S-arc can only be expanded in the presence of an NP, we also add arcs for the relevant NP-predictions.



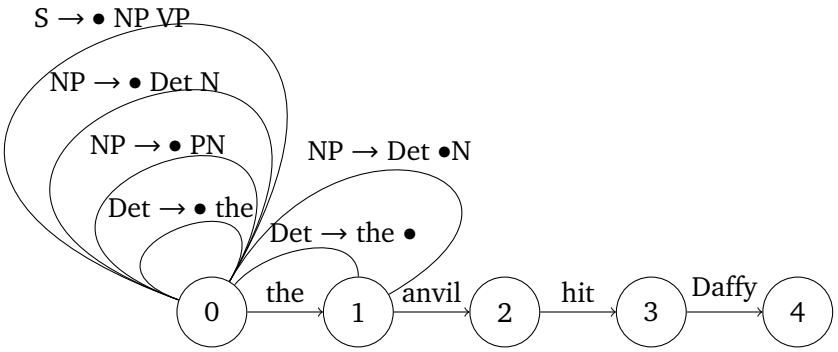
Following the same line of reasoning, we also have to add branches for each possible rewriting of Det and PN. We only add one Det-arc here to avoid cluttering the figure.



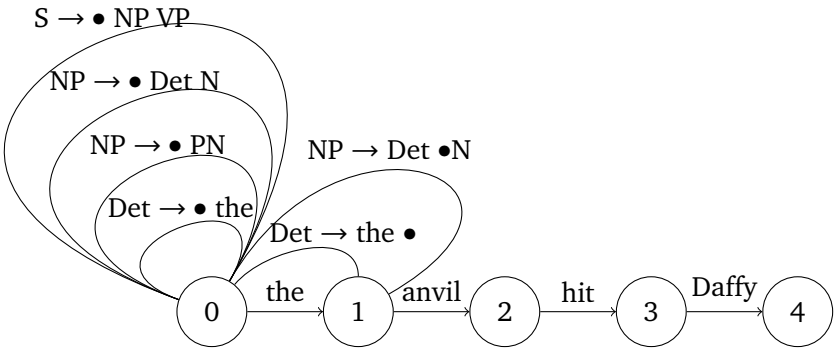
Now that we have an arc that can be expanded if we have the right terminal symbol, the parser reads in the word starting at position 0, which is *the*. This means that we have confirmed previously predicted information of the Det-arc in position 0, so that we also add a completed Det-arc from 0 to 1.



We can now combine this arc with the NP → •Det N arc from 0 to 0 to produce a new arc NP → Det •N from 0 to 1.



This arc, in turn is used to add arcs from 1 to 1 that are labeled with dotted rewrite rules for N, e.g. N → •anvil. We add these arcs because the NP-arc from 0 to 1 can only be completed if there is an N-arc starting at position 1.



Exercise 8.1. Continue the example parse until you have reached the S-arc. ⊙

Exercise 8.2. Use this arc-drawing system to parse the sentence *then anvil hit the duck on the head* with the expanded toy grammar we used for the CKY parser. Does this format offer a way of representing the structural ambiguity? ⊙

2 Formal Specification

2.1 Parsing Schema

In its arc-based representation the Earley parser may look rather unusual to you, but the item-based parsing schema should strike you as remarkably familiar. Items are of the form $[i, N \rightarrow \alpha \bullet \beta, j]$ and indicate that an N-constituent starts in position i and has been recognized up to position j. As our axiom we pick $[0, \mathrm{S}' \rightarrow \bullet \mathrm{S}, 0]$. This is just a convenient shorthand so that we do not have to posit an axiom for every rewrite rule for S in our grammar. Given this convention, then goal item is $[0, \mathrm{S}' \rightarrow \mathrm{S}\bullet, n]$, of course.

The inference rules are variants of rules that we already encountered with the shift-reduce parser and the LC parser. Scan turns a predicted terminal symbol into recognized information (so it is the counterpart of the LC parser's shift rule rather than the scan rule, which is used to cancel out predicted and confirmed information).

$$\textbf{Scan} \qquad \frac{[i, A \rightarrow \alpha \bullet a\beta, j]}{[i, A \rightarrow \alpha a \bullet \beta, j+1]}\ a = w_i$$

Predict is similar to standard top-down prediction. But just like the CKY algorithm breaks large items into smaller ones that can be easily stored in a chart and used in several inference steps, the Earley parser predicts much smaller items that explicitly represent conjectured subtrees.

$$\textbf{Predict} \qquad \frac{[i, A \rightarrow \alpha \bullet B\beta, j]}{[j, B \rightarrow \bullet\gamma, j]}\ B \rightarrow \gamma \in R$$

The complete rule, finally, mirrors the behavior of the LC parser's complete rule in that it turns confirmed predictions into recognized material. The only difference is that now the combination of two parse items is needed to license this inference.

$$\textbf{Complete} \qquad \frac{[i, A \rightarrow \alpha \bullet B\beta, j] \qquad [j, B \rightarrow \gamma\bullet, k]}{[i, A \rightarrow \alpha B \bullet \beta, k]}$$

Exactly like the CKY parser, the Earley parser is mostly used as a fully parallel chart parser that explores all possible parses at the same. The Earley parsing schema, however, is independent of such considerations and can also be applied in a purely serial fashion. In this case the Earley parser behaves similar to a recursive-descent parser.

Example 8.1 Parse Table of an Earley Parse

Consider once more the sentence *the anvil hit Daffy*, parse with our usual toy grammar. Then the shortest possible sequential Earley parse yields a parse table that closely resembles the one of the recursive-descent parser.

parse item	inference rule	parse item	inference rule
[0, •S, 4]	axiom	[0, S′ →•S, 0]	axiom
[0, •NP VP, 4]	predict(1)	[0, S →•NP VP, 0]	predict(1)
[0, •Det N VP, 4]	predict(3)	[0, NP →•Det N, 0]	predict(3)
[0, •the N VP, 4]	predict(6)	[0, Det →•the, 0]	predict(6)
[1, •N VP, 4]	scan	[0, Det →the •, 1]	scan
		[0, NP →Det •N, 1]	complete
[1, •truck VP, 4]	predict(7)	[1, N →•truck, 1]	predict(7)
[2, •VP, 4]	scan	[1, N →truck •, 2]	scan
		[0, NP →Det N •, 2]	complete
		[0, S →NP •VP, 2]	complete
[2, •Vt NP, 4]	predict(5)	[2, VP →•Vt NP, 2]	predict(5)
[2, •hit NP, 4]	predict(10)	[2, Vt →•hit, 2]	predict(10)
[3, •NP, 4]	scan	[2, Vt →hit •, 3]	scan
		[2, VP →Vt •NP, 3]	complete
[3, •PN, 4]	predict(2)	[3, NP →•PN, 3]	predict(2)
[3, •Daffy, 4]	predict(8)	[3, PN →•Daffy, 3]	predict(8)
[4, •, 4]	scan	[3, PN →Daffy •, 4]	scan
		[3, NP →PN •, 4]	complete
		[2, VP →Vt NP •, 4]	complete
		[0, S →NP VP •, 4]	complete
		[0, S′ →S•, 4]	complete

The direct comparison of parse tables in the example above highlights that we may think of the recursive-descent parser as a particular instantiation of the Earley parser: a single recursive-descent scan represents an Earley scan followed by one or more completion steps. In general, a parser that takes fewer steps is more efficient than one that takes more steps. So if we are interested in designing a serial parser with backtracking that does not reuse parse items, recursive-descent is a better choice than Earley. In other words, recursive-descent is a smart modification of the Earley parsing schema for the purpose of serial parsing.

This view also highlights the downside of recursive-descent in comparison to Earley: the latter can easily reuse parse items and thus is a much more natural candidate whenever the parser is allowed to memorize and reuse parse items. This makes Earley a much better choice for chart parsing and industrial applications.

2.2 Using a Chart

3 Left Recursion is Unproblematic

4 Relation to Left Corner Parsing

4.1 Removing Top-Down Filtering

4.2 Left Corner Parsing as a Step Contraction