Lecture 9

Schema Refinement and Filtering

In this chapter we will look at several techniques for converting one parsing schema into another: item refinement, step refinement, static filtering, dynamic filtering, and step contraction. The latter we already encountered in Chap. 6, where we observed that I) the LC reduce rule is a step contraction of predict followed by complete, and II) the scan rule of recursive descent parsing schema can be viewed as a step contraction of LC shift followed by LC scan. Step contraction and the other mechanisms allow us to establish surprising relations between seemingly incomparable parsing schema. This is not only of theoretical interest. Since many of the techniques can be viewed as abstractions of algorithmic speed-up techniques (e.g. precompilation, run-time optimization), they also explain why some parsers are faster than others.

1 A Base Case: Bottom-Up Earley

As a useful base case that will simplify comparisons later on, we define a simplified variant of the Earley parser without the predict rule. The only valid inference rules thus are scan and complete.

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

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

The goal item is still $[0, S' \to S\bullet, n]$. The axiom, on the other hand, has to be changed since the absence of a predict rule means that all items must be inferred bottom-up. We do so by allowing every prediction to take place at every position in the input. That is to say, for every position i and rewrite rule $A \to \alpha$ of the grammar, there is a corresponding axiom $[i, A \to \bullet\alpha, i]$.

Exercise 9.1. Construct the chart for a simplified Earley parse of *the anvil hit the duck on the head* using the extended toy grammar from Chap. 7. Feel free to omit some of the axioms, which would require a lot of ink to be spilled. ⊙

Before moving on, we can already note two particular insights. First, the simplified Earley parser is a pure bottom-up parser, similar to shift-reduce or CKY. Second,

standard Earley parsing constructs fewer arcs than simplified Earley parsing. So the former is the result of enriching the latter with top-down filtering on predicted items. The relation between the two parsing schemata is similar to the relation that holds between the standard LC parsing schema and the extended variant with top-down filtering (cf. Sec. 2.2 of Cha. 6). This also highlights that even though the Earley parser intuitively feels like a top-down parser, it is best viewed on a formal level as a bottom-up parser (which also explains immediately why left recursion is unproblematic). To further emphasize this point, we will refer to the simplified Earley schema as bottom-up Earley (buE).

2 Refinement

2.1 Defining Item and Step Refinements

We now introduce our first two tools for relating parsers: item refinement and step refinement. Both are fairly easy to understand on an intuitive level. A parsing schema P is an *item refinement* of schema Q if the items of Q are more fine-grained. Technically, this means that we can define a total surjective map from Q's parse items to P's parse items such that the image of Q's parses under this map is exactly the parses of P.

total A function f from A to B is *total* iff $f(a)$ is defined for every $a \in A$.

surjective A function f from A to B is *surjective* iff for every $b \in B$ there is an $a \in A$ such that $f(a) = b$. In this case f is also said to be *onto*.

Step refinement applies the same concept to sequences of inference rules. A parsing schema P is a *step refinement* of schema Q iff for every inference rule of Q there is an equivalent sequence of one or more inference steps in P. If this still seems awfully abstract enough, don't worry, we will work through a concrete example next.

2.2 Item Refinement of CKY

Recall that CKY uses items of the form $[i, A, j]$, which are operated on by two rules:

$$\textbf{Shift} \qquad \frac{}{[i, A, i+1]} \; A \rightarrow a, a = w_i$$

$$\textbf{Reduce} \qquad \frac{[i, B, j] \qquad [j, C, k]}{[i, A, k]} \; A \rightarrow BC \in R$$

We will now refine CKY into a new parsing schema CKY$^{\text{E}}$ that uses Earley-style productions instead of just non-terminals. So items are of the general form $[i, A \rightarrow \alpha, j]$, assuming that $A \rightarrow \alpha$ is a valid rewrite rule of a context-free grammar in Chomsky Normal Form. It is a trivial matter to adapt the inference rules accordingly:

$$\textbf{Shift} \qquad \frac{}{[i, A \rightarrow a, i+1]} \; A \rightarrow a, a = w_i$$

$$\textbf{Reduce} \qquad \frac{[i, B \rightarrow \beta, j] \qquad [j, C \rightarrow \gamma, k]}{[i, A \rightarrow BC, k]} \; A \rightarrow BC \in R$$

The relation between parsing items is equally obvious. Every parse item $[i,A \rightarrow \alpha, j]$ of CKY$^{\text{E}}$ is mapped to $[i,A,j]$ in CKY. Whatever parse of CKY$^{\text{E}}$ this function is applied to, it yields a parse of CKY. Furthermore, every parse of CKY is the image of some CKY$^{\text{E}}$ parse under this function. These three facts jointly establish that CKY$_{\text{E}}$ is an item refinement of CKY.

2.3 Bottom-Up Earley as a Refinement of Generalized CKY

Considered in isolation, CKY$^{\text{E}}$ is rather unremarkable as it only adds unnecessary redundancy to the standard CKY schema. But in the grand scheme of things, it is a useful step towards relating CKY and Earley parsing. Towards this end, we now define another schema ECKY and show that it is a step refinement of CKY$^{\text{E}}$. ECKY modifies item format as CKY$^{\text{E}}$ to proper Earley-style dotted productions, i.e. $[i,A \rightarrow \alpha \bullet \beta, i]$. The inference rules are also modified.

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

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

In addition, we posit an axiom $[i,A \rightarrow \bullet\alpha, i]$ for every input position i and rewrite rule $A \rightarrow \alpha$ of the grammar. You might point out that ECKY is identical to buE, but this is only partially correct. Since ECKY is only defined for grammars in Chomsky Normal Form, it is a special case of buE, which works with arbitrary grammars. The difference between the two is obscured by our use of variables like α, β and γ. Every licit instantiation of those variables for ECKY is also valid for buE, but not the other way round. Still, the point stands that ECKY is essentially buE restricted to grammars in Chomsky Normal Form.

In order to show that ECKY is a step refinement of CKY$^{\text{E}}$, we exhibit equivalent sequences of inference steps for CKY$^{\text{E}}$'s shift and reduce rules. This presupposes that we already know how to relate parse items of CKY$^{\text{E}}$ to ECKY. Fortunately this is easy in this case: every item $[i,A \rightarrow \alpha, j]$ of CKY$_{\text{E}}$ is equated with $[i,A \rightarrow \alpha\bullet, j]$ of ECKY. Now suppose that we infer $[i,A \rightarrow a, i+1]$ via the shift rule of CKY$^{\text{E}}$. ECKY instead uses the axiom $[i,A \rightarrow \bullet a, i]$ and the scan rule to infer $[i,A \rightarrow a\bullet, i+1]$, which is the ECKY correspondent of CKY$^{\text{E}}$'s $[i,A \rightarrow a, i+1]$. Where CKY$^{\text{E}}$ uses its reduce rule to infer $[i,A \rightarrow BC, k]$ from $[i,B \rightarrow \beta, j]$ and $[j,C \rightarrow \gamma, k]$, ECKY instead invokes two complete steps:

$$\frac{\dfrac{[i,A \rightarrow \bullet BC, i] \qquad [i,B \rightarrow B\bullet, j]}{[i,A \rightarrow B \bullet C, j]} \qquad [j,C \rightarrow \gamma\bullet, k]}{[i,A \rightarrow BC\bullet, j]}$$

As $[i,A \rightarrow BC\bullet, k]$ corresponds to $[i,A \rightarrow BC, k]$, ECKY is indeed a step refinement of CKY$^{\text{E}}$.

At this point, we know that ECKY is a step refinement of CKY$^{\text{E}}$, which in turn is an item refinement of CKY. For the sake of simplicity, any sequence of refinement steps is called a *refinement*. So ECKY is a *refinement* of CKY.

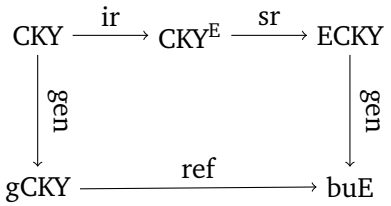
This result may still strike you as fairly artificial since neither ECKY nor CKY$^{\text{E}}$ have been proposed independently in the parsing literature. The only thing we have

accomplished is to relate parsing schema that we made up little more than a page ago, seemingly on a whim. But remember that ECKY is a special case of buE. The two behave exactly the same except that ECKY is only defined for grammars in Chomsky Normal Form. But of course the steps we took to refine CKY into ECKY can also be repeated with generalized CKY (gCKY).

Exercise 9.2. Guess what? You get to do just that. ⊙

The kind of generalized ECKY that is obtained this way is exactly buE. Consequently, buE is a refinement of gCKY. Or the other way round, gCKY is a more concise variant of buE that simplifies items and compacts certain inference sequences into a single inference step.

Our findings so far can be summarized in a simple graph. Here parsing schema P is considered a generalization of Q iff P is defined for a superset of the grammars Q is defined for and the two behave exactly the same over their shared grammars. The relations item refinement, step refinement, refinement and generalization are abbreviated as ir, sr, ref, and gen, respectively.



3 Filtering

3.1 Defining Static and Dynamic Filtering

Whereas refinement "grows" a parsing schema by adding new items and/or decomposing single inference steps into a sequence thereof, filtering is all about making parsing schemata more compact. Filters are in particular meant to reduce the amount of work that needs to be done during the construction of a parse. The simplest option is a *static filter*, which simply eliminates certain items or inference rules. A parsing schema P is a static filtering of Q ($Q \xrightarrow{sf} P$) iff I) all parse items of P are parse items of Q, and II) every inference rule of P is an inference rule of Q.

Dynamic filters are of greater interest for our purposes. Whereas static filters discard parts of the parsing schema to simplify the construction of parses, dynamic filters minimize workload by putting additional restrictions on the applicability of inference rules. Technically, parsing schema P is a dynamic filtering of schema Q ($Q \xrightarrow{df} P$) iff I) all items of P are items of Q, and II) all valid inference steps of P are valid inference steps of Q. Notice that this definition makes static filters a special case of dynamic filters. Whereas static filters are more blunt and remove entire inference rules, dynamic filters can retain all inference rules but restrict their applicability.

3.2 Earley is a Dynamic Filtering of Bottom-Up Earley

We have already encountered an instance of dynamic filtering: in Chap. 6, we added top-down filtering to the standard left-corner parser. This mechanism preserved all

parse items and inference rules (so it is not a static filtering), but it limited the reduce rule so that only valid left corners could be inferred. For another example, remember that buE was obtained from Earley by removing exactly the kind of top-down filtering that we added to the left-corner parser. It stands to reason, then, that buE is a dynamic filtering of Earley — that is indeed the case.

Note first that buE and Earley have exactly the same set of valid items. They also have exactly the same scan and complete rules. The main differences thus lie in the predict rule and the choice of axioms. Given the definition of dynamic filtering, it suffices to show that no licit inference step of Earley is illict in buE. So suppose we predict $[j, B \to \bullet\gamma, j]$ from $[i, A \to \alpha \bullet B\beta, j]$. Then $[j, B \to \bullet\gamma, j]$ is a valid inference in buE because it is one of its axioms. The same holds for the Earley parser's single axiom, $[0, S' \to \bullet S, 0]$.

Exercise 9.3. Show that the LC parsing schema with top-down filtering is a dynamic filtering of the standard LC schema. ⊙

3.3 Defining Step Contraction

The most powerful type of filtering is *step contraction*, which — somewhat surprisingly — is just the inverse of step refinement. That is to say, parsing schema P is a step contraction of schema Q ($Q \xrightarrow{sc} P$) iff Q is a step refinement of P. Why do we need both step contraction and step refinement if one is just the inverse of the other? Because both step contraction and step refinement have their uses. In particular, both can lead to significant speed-ups in the parser (the flip side being that both can also make the parser slower).

It is also interesting that every dynamic filter is a step contraction, and every inverse step contraction (i.e. a refinement) is a static filter. This means we have the following subsumption relation: ref $\subseteq$ sf $\subseteq$ df $\subseteq$ sc.

Exercise 9.4. Explain why ref $\subseteq$ sf and df $\subseteq$ sc hold. ⊙

3.4 Left Corner Parsing is a Step Contraction of Earley

For the sake of convenience, we redefine the parsing schema for the left-corner parser with top-down filtering to bring it more in line with the Earley schema. Instead of the complicated LC parse items, we follow the format $[i, A \to \alpha \bullet \beta, j]$ of the Earley parser. We also adopt the axiom of the Earley parser: $[0, S' \to \bullet S, 0]$.

The rewrite rules also undergo some major modifications. Remember that the reduce rule can be emulated by predict followed by complete, so we drop it. This leaves us with shift, scan, predict and complete. We recombine shift and scan into the more standard scan rule as it is familiar from the recursive-descent and Earley parsers. We then split predict into two rules depending on whether it is triggered by a terminal symbol or a non-terminal. Complete is just the standard complete rule of the Earley parser. So the full set of rewrite rules looks as follows:

Scan
$$\frac{[i, A \to \alpha \bullet a\beta, j]}{[i, A \to \alpha a \bullet \beta, j+1]}\ w_j = a$$

Predict Non-Terminal
$$\frac{[i, C \to \gamma \bullet E\delta, j] \qquad [j, A \to \alpha\bullet, k]}{[j, B \to A \bullet \beta, k]}\ B \in \mathrm{lc}(E)$$

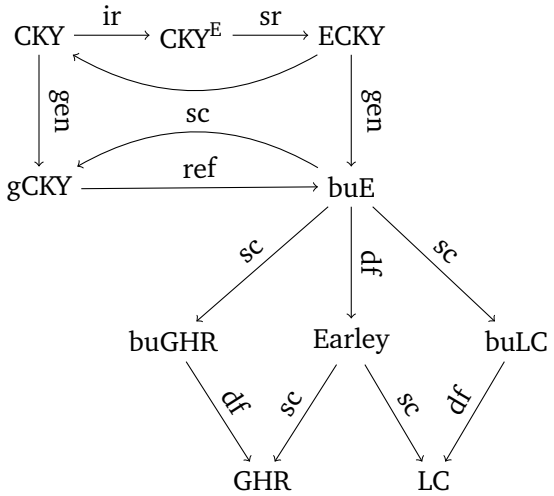
Predict Terminal $\dfrac{[i, C \to \gamma \bullet E\delta, j]}{[j, B \to a \bullet \beta, k]}\ a = w_j, B \in \mathrm{lc}(E)$

Complete $\dfrac{[i, A \to \alpha \bullet B\beta, j] \qquad [j, B \to \gamma, k]}{[i, A \to \alpha B \bullet \beta, k]}$

Exercise 9.5. Verify for yourself that this parser still builds trees in the same fashion as our original left-corner parser with top-down filtering. You might want to start by constructing the parse for our usual example sentence *The anvil hit Daffy*, and possibly *John's father's car's exhaust pipe disappeared.* ⊙

With these changes in the parsing schema it is very easy to recognize the LC parser with top-down filtering as a step contraction of the Earley parser. To prove it, we have to show that the Earley parser is a refinement of the LC parser. Every LC parse item is also an Earley item, so Earley is an item refinement of LC. In addition, Earley is also a step refinement. The scan and complete rule are identical between the two. As we establish next, Predict Non-Terminal and Predict-Terminal correspond to sequences of Earley predict and completion steps.

Consider the LC parser's Predict Non-Terminal rule. It allows the LC parser to infer $[j, B \to A \bullet \beta, k]$ provided $[i, C \to \gamma \bullet E\delta, j]$ and $[j, A \to \alpha\bullet, k]$ are valid. Now suppose that we are given $[i, C \to \gamma \bullet E\delta, j]$ in the Earley parser. Since B must be a left-corner of E, some finite sequence of predict steps infers $[j, B \to \bullet A\beta, j]$ from $[i, C \to \gamma \bullet E\delta, j]$. From this we infer $[j, A \to \bullet\alpha, j]$, which is eventually completed as $[j, A \to \alpha\bullet, k]$. Another complete step then produces $[j, B \to A \bullet \beta, k]$. A similar argument also works for Predict Terminal.



Exercise 9.6. Show that buE $\xrightarrow{sc}$ buGHR and buGHR $\xrightarrow{df}$ GHR hold. ⊙

Exercise 9.7. The diagram also shows relations for the GHR parsing schema and a bottom-up version without top-down filtering. Give a definition of buGHR, then prove the relevant relations. ⊙

Exercise 9.8. How could the generalized left-corner parser — and by extension the recursive-descent and shift-reduce parsers — be fit into the diagram? ⊙

4 Psycholinguistic Connections

I frankly don't know. That would be an interesting issue to explore.