Efficient Semiring-Weighted Earley Parsing

Andreas Opedal[1,2] **Ran Zmigrod**[3] **Tim Vieira**[4]
Ryan Cotterell[1] **Jason Eisner**[4]
[1]ETH Zürich [2]Max Planck ETH Center for Learning Systems
[3]University of Cambridge [4]Johns Hopkins University
andreas.opedal@inf.ethz.ch rz279@cam.ac.uk tim.f.vieira@gmail.com
ryan.cotterell@inf.ethz.ch jason@cs.jhu.edu

Abstract

This paper provides a reference description, in the form of a deduction system, of Earley's (1970) context-free parsing algorithm with various speed-ups. Our presentation includes a known worst-case runtime improvement from Earley's $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$, which is unworkable for the large grammars that arise in natural language processing, to $\mathcal{O}\big(N^3|\mathcal{G}|\big)$, which matches the runtime of CKY on a binarized version of the grammar $\mathcal{G}$. Here N is the length of the sentence, $|\mathcal{R}|$ is the number of productions in $\mathcal{G}$, and $|\mathcal{G}|$ is the total length of those productions. We also provide a version that achieves runtime of $\mathcal{O}\big(N^3|\mathcal{M}|\big)$ with $|\mathcal{M}| \leq |\mathcal{G}|$ when the grammar is represented compactly as a single finite-state automaton $\mathcal{M}$ (this is partly novel). We carefully treat the generalization to semiring-weighted deduction, preprocessing the grammar like Stolcke (1995) to eliminate deduction cycles, and further generalize Stolcke's method to compute the weights of sentence prefixes. We also provide implementation details for efficient execution, ensuring that on a preprocessed grammar, the semiring-weighted versions of our methods have the same asymptotic runtime and space requirements as the unweighted methods, including sub-cubic runtime on some grammars.

https://github.com/rycolab/earleys-algo

1 Introduction

Earley (1970) was a landmark paper in computer science.[1] Its algorithm was the first to directly parse under an *unrestricted* context-free grammar in time $\mathcal{O}\big(N^3\big)$, with N being the length of the input string. Furthermore, it is faster for certain grammars because it uses left context to filter its search at each position. It parses unambiguous grammars in $\mathcal{O}\big(N^2\big)$ time and a class of "bounded-state" grammars, which includes all deterministic grammars, in $\mathcal{O}(N)$ time. Its artful combination of top-down (goal-driven) and bottom-up (data-driven) inference later inspired a general method for executing logic programs, "Earley deduction" (Pereira and Warren, 1983).

Earley's algorithm parses a sentence incrementally from left to right, optionally maintaining a packed parse forest over the sentence prefix that has been observed so far. This supports online sentence processing—incremental computation of syntactic features and semantic interpretations—and also reveals for each prefix the set of grammatical choices for the next word.[2]

It can be attractively extended to compute the *probabilities* of the possible next words (Jelinek and Lafferty, 1991; Stolcke, 1995). This is a standard way to compute autoregressive language model probabilities under a PCFG to support cognitive modeling (Hale, 2001) and speech recognition (Roark, 2001). Such probabilities could further be combined with those of a large autoregressive language model to form a product-of-experts model. Recent papers (as well as multiple github projects) have made use of a restricted version of this, restricting generation from the language model to only extend the current prefix in ways that are grammatical under an unweighted CFG; then only grammatical text or code will be generated (Shin et al., 2021; Roy et al., 2022; Fang et al., 2023).

It is somewhat tricky to implement Earley's algorithm so that it runs as fast as possible. Most importantly, the worst-case runtime should be linear in the size of the grammar, but this property was not achieved by Earley (1970) himself nor by textbook treatments of his algorithm (e.g., Jurafsky and Martin, 2009, §13.4). This is easy to overlook when the grammar is taken to be fixed, so that the grammar constant is absorbed into the $\mathcal{O}$ operator, as in the opening paragraph of this paper.

[1]Based on the author's dissertation (Earley, 1968).

[2]In a programming language editor, incremental interpretation can support syntax checking, syntax highlighting, and tooltips; next-word prediction can support autocomplete.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 3687–3713

Yet reducing the grammar constant is critical in practice, since natural language grammars can be very large (Dunlop et al., 2010). For example, the Berkeley grammar (Petrov et al., 2006), a learned grammar for the Penn Treebank (PTB) (Marcus et al., 1993), contains over one million productions.

In this reference paper, we attempt to collect the key efficiency tricks and present them declaratively, in the form of a unified *deduction system* that can be executed with good asymptotic complexity.[3] We obtain further speedups by allowing the grammar to be presented in the form of a weighted finite-state automaton whose paths correspond to the productions, which allows similar productions to share structure and thus to share computation. Previous versions of this trick use a different automaton for each left-hand side nonterminal (Purdom and Brown, 1981; Kochut, 1983; Leermakers, 1989; Perlin, 1991, *inter alia*); we show how to use a single automaton, which allows further sharing among productions with different left-hand sides.

We carefully generalize our methods to handle semiring-weighted grammars, where the parser must compute the total weight of all trees that are consistent with an observed sentence (Goodman, 1999)—or more generally, consistent with the prefix that has been observed so far. Our goal is to ensure that if the semiring operations run in constant time, then semiring-weighted parsing runs in the same time and space as unweighted parsing (up to a constant factor), for *every* grammar and sentence, including those where unweighted parsing is faster than the worst case. Eisner (2023) shows how to achieve this guarantee for any *acyclic* deduction system, so we produce such a system by preprocessing the grammar to eliminate cyclic derivations.[4] Intuitively, this means we do not have to sum over infinitely many derivations at runtime (as Goodman (1999) would). We also show how to compute prefix weights, which is surprisingly tricky and requires the semiring to be commutative. Our presentation of preprocessing and prefix weights generalizes and corrects that of Stolcke (1995), who relied on special properties of PCFGs.

Finally, we provide a reference implementation in Cython[5] and empirically demonstrate the value of the speedups.

[3]There has been no previous unified, formal treatment that is written as a deduction system, to the best of our knowledge. That said, declarative formulations have been presented in other formats in the dissertations of Barthélemy (1993), de la Clergerie (1993), and Nederhof (1994a).

[4]Our method to remove nullary productions may be a contribution of this paper, as we were unable to find a correct construction in the literature.

2 Weighted Context-Free Grammars

A **context-free grammar** (**CFG**) $\mathcal{G}$ is a tuple $\langle \mathcal{N}, \Sigma, \mathcal{R}, S \rangle$ where Σ is a finite set of **terminal** symbols, $\mathcal{N}$ is a finite set of **nonterminal** symbols with $\Sigma \cap \mathcal{N} = \emptyset$, $\mathcal{R}$ is a set of **productions** from a nonterminal to a sequence of terminals and nonterminals (i.e., $\mathcal{R} \subseteq \mathcal{N} \times (\Sigma \cup \mathcal{N})^*$), and $S \in \mathcal{N}$ is the **start** symbol. We use lowercase variable names ($a, b, \ldots$) and uppercase ones ($A, B, \ldots$) for elements of Σ and $\mathcal{N}$, respectively. We use a Greek letter (ρ, μ, or ν) to denote a sequence of terminals and nonterminals, i.e., an element of $(\Sigma \cup \mathcal{N})^*$. Therefore, a production has the form $A \to \rho$. Note that ρ may be the empty sequence ε. We refer to $|\rho| \geq 0$ as the **arity** of the production, $|A \to \rho| \stackrel{\text{def}}{=} 1 + |\rho|$ as the **size** of the production, and $|\mathcal{G}| \stackrel{\text{def}}{=} \sum_{(A \to \rho) \in \mathcal{R}} |A \to \rho|$ for the total **size** of the CFG. Therefore, if K is the maximum arity of a production, $|\mathcal{G}| \leq |\mathcal{R}|(1 + K)$. Productions of arity 0, 1, and 2 are referred to as **nullary**, **unary**, and **binary** productions respectively.

For a given $\mathcal{G}$, we write $\mu \Rightarrow \nu$ to mean that $\mu \in (\Sigma \cup \mathcal{N})^*$ can be rewritten into $\nu \in (\Sigma \cup \mathcal{N})^*$ by a single production of $\mathcal{G}$. For example, $A\,B \Rightarrow \rho\,B$ expands A into ρ using the production $A \to \rho$. The reflexive and transitive closure of this relation, $\stackrel{*}{\Rightarrow}$, then denotes rewriting by any sequence of zero or more productions: for example, $A\,B \stackrel{*}{\Rightarrow} \rho\,\mu\,\nu$. We may additionally write $A \stackrel{*}{\Rightarrow}_L \rho$ iff $A \stackrel{*}{\Rightarrow} \rho\,\mu$, and refer to ρ as a **prefix** of $\rho\,\mu$.

A **derivation subtree** of $\mathcal{G}$ is a finite rooted ordered tree T such that each node is labeled either with a terminal $a \in \Sigma$, in which case it must be a leaf, or with a nonterminal $A \in \mathcal{N}$, in which case $\mathcal{R}$ must contain the production $A \to \rho$ where ρ is the sequence of labels on the node's 0 or more children. For any $A \in \mathcal{N}$, we write $\mathcal{T}^A$ for the set of derivation subtrees whose roots have label A, and refer to the elements of $\mathcal{T}^S$ as **derivation trees**. Given a string $\mathbf{x} \in \Sigma^*$ of length N, we write $\mathcal{T}^A_{\mathbf{x}}$ for the set of derivation subtrees with leaf sequence $\mathbf{x}$. For an **input sentence** $\mathbf{x}$, its set of derivation trees $\mathcal{T}_{\mathbf{x}} \stackrel{\text{def}}{=} \mathcal{T}^S_{\mathbf{x}}$ is countable and possibly infinite. It is non-empty iff $S \stackrel{*}{\Rightarrow} \mathbf{x}$, with each $T \in \mathcal{T}_{\mathbf{x}}$ serving as a witness that $S \stackrel{*}{\Rightarrow} \mathbf{x}$, i.e., that $\mathcal{G}$ can generate $\mathbf{x}$.

We will also consider **weighted CFG**s

[5]A fast implementation of Earley's algorithm is reported by Polat et al. (2016) but does not appear to be public.

	`Earley`	`EarleyFast`
Domains	$i,j,k \in \{0,\ldots,N\} \quad A,B \in \mathcal{N} \cup \{S'\} \quad a \in \Sigma \quad \rho,\mu,\nu \in (\mathcal{N} \cup \Sigma)^*$	
Items	$[i,j,A \to \mu \bullet \nu] \quad [j,k,a] \quad A \to \rho$	$[i,j,A \to \mu \bullet \nu] \quad [j,k,a] \quad A \to \rho$ $[i,j,A \to \bullet \star] \quad [i,j,A \to \star \bullet]$
Axioms	$A \to \rho, \forall (A \to \rho) \in \mathcal{R}$ $[k-1,k,x_k], \forall k \in \{1,\ldots,N\}$ $[0,0,S' \to \bullet S]$	$A \to \rho, \forall (A \to \rho) \in \mathcal{R}$ $[k-1,k,x_k], \forall k \in \{1,\ldots,N\}$ $[0,0,S \to \bullet \star]$
Goal	$[0,N,S' \to S \bullet]$	$[0,N,S \to \star \bullet]$
Rules	PRED: $\dfrac{B \to \rho}{[j,j,B \to \bullet \rho]}\ [i,j,A \to \mu \bullet B\,\nu]$	PRED1: $\dfrac{}{[j,j,B \to \bullet \star]}\ [i,j,A \to \mu \bullet B\,\nu]$ PRED2: $\dfrac{B \to \rho}{[j,j,B \to \bullet \rho]}\ [j,j,B \to \bullet \star]$
	SCAN: $\dfrac{[i,j,A \to \mu \bullet a\,\nu] \quad [j,k,a]}{[i,k,A \to \mu\,a \bullet \nu]}$	SCAN: $\dfrac{[i,j,A \to \mu \bullet a\,\nu] \quad [j,k,a]}{[i,k,A \to \mu\,a \bullet \nu]}$ COMP1: $\dfrac{[j,k,B \to \rho \bullet]}{[j,k,B \to \star \bullet]}$
	COMP: $\dfrac{[i,j,A \to \mu \bullet B\,\nu] \quad [j,k,B \to \rho \bullet]}{[i,k,A \to \mu\,B \bullet \nu]}$	COMP2: $\dfrac{[i,j,A \to \mu \bullet B\,\nu] \quad [j,k,B \to \star \bullet]}{[i,k,A \to \mu\,B \bullet \nu]}$

Table 1: Deduction systems for Earley (1970)'s algorithm (`Earley`) and our faster algorithm (`EarleyFast`). An additional speedup is given in App. G.2. Properly speaking, the "items" and "rules" shown here are templates; the actual items and rules are obtained by binding their variables to elements of their corresponding domains.

(**WCFG**s), in which each production $A \to \rho$ is additionally equipped with a **weight** $w(A \to \rho) \in \mathbb{W}$ where $\mathbb{W}$ is the set of values of a **semiring** $\mathcal{S} \stackrel{\text{def}}{=} \langle \mathbb{W}, \oplus, \otimes, \mathbb{0}, \mathbb{1} \rangle$. Semirings are defined in App. A. We assume that $\otimes$ is commutative, deferring the trickier non-commutative case to App. K. Any derivation tree T of $\mathcal{G}$ can now be given a weight

$$w(T) \stackrel{\text{def}}{=} \bigotimes_{(A \to \rho) \in T} w(A \to \rho) \tag{1}$$

where $A \to \rho$ ranges over the productions associated with the nonterminal nodes of T. The goal of a **weighted recognizer** is to find the total weight of all derivation trees of a given input sentence $\mathbf{x}$:

$$\mathrm{Z}_{\mathbf{x}} \stackrel{\text{def}}{=} w\left(S \stackrel{*}{\Rightarrow} \mathbf{x}\right) \stackrel{\text{def}}{=} \bigoplus_{T \in \mathcal{T}_{\mathbf{x}}} w(T) \tag{2}$$

An ordinary **unweighted recognizer** is the special case where $\mathbb{W}$ is the boolean semiring, so $\mathrm{Z}_{\mathbf{x}} = \mathrm{true}$ iff $S \stackrel{*}{\Rightarrow} \mathbf{x}$ iff $\mathcal{T}_{\mathbf{x}} \neq \emptyset$. A **parser** returns at least one derivation tree from $\mathcal{T}_{\mathbf{x}}$ iff $\mathcal{T}_{\mathbf{x}} \neq \emptyset$.

As an extension to the weighted recognition problem (2), one may wish to find the **prefix weight** of a string $\mathbf{y} \in \Sigma^*$, which is the total weight of all sentences $\mathbf{x} = \mathbf{yz} \in \Sigma^*$ having that prefix:

$$w\left(S \stackrel{*}{\Rightarrow}_L \mathbf{y}\right) \stackrel{\text{def}}{=} \bigoplus_{\mathbf{z} \in \Sigma^*} w\left(S \stackrel{*}{\Rightarrow} \mathbf{yz}\right) \tag{3}$$

§1 discussed applications of **prefix probabilities**—the special case of (3) for a **probabilistic CFG** (**PCFG**), in which the production weights are rewrite probabilities: $\mathbb{W} = \mathbb{R}_{\geq 0}$ and $(\forall A \in \mathcal{N}) \sum_{(A \to \rho) \in \mathcal{R}} w(A \to \rho) = 1$.

3 Parsing as Deduction

We will describe Earley's algorithm using a **deduction system**, a formalism that is often employed in the presentation of parsing algorithms (Pereira and Shieber, 1987; Sikkel, 1997), as well as in mathematical logic and programming language theory (Pierce, 2002). Much is known about how to execute (Goodman, 1999), transform (Eisner and Blatz, 2007), and neuralize (Mei et al., 2020) deduction systems.

A deduction system proves **items** V using **deduction rules**. Items represent propositions; the rules are used to prove all propositions that are true. A deduction rule is of the form

$$\textsc{Example:}\ \frac{U_1 \qquad U_2 \qquad \cdots}{V}$$

where EXAMPLE is the name of the rule, the 0 or more items above the bar are called **antecedents**, and the single item below the bar is called a **consequent**. Antecedents may also be written to the side of the bar; these are called **side conditions** and will be handled differently for weighted deduction in §6. **Axioms** (listed separately) are merely rules that have no antecedents; as a shorthand, we omit the bar in this case and simply write the consequent.

A **proof tree** is a finite rooted ordered tree whose nodes are labeled with items, and where every node is licensed by the existence of a deduction rule whose consequent V matches the label of the node and whose antecedents $U_1, U_2, \ldots$ match the labels of the node's children. It follows that the leaves are labeled with axioms. A **proof** of item V is a proof tree d_V whose root is labeled with V: this shows how V can be deduced from its children, which can be deduced from their children, and so on until axioms are encountered at the leaves. We say V is **provable** if $\mathcal{D}_V$, which denotes the set of all its proofs, is nonempty.

Our unweighted recognizer determines whether a certain **goal item** is provable by a certain set of deduction rules from axioms that encode $\mathcal{G}$ and $\mathbf{x}$. The deduction system is set up so that this is the case iff $S \overset{*}{\Rightarrow} \mathbf{x}$. The recognizer can employ a **forward chaining** method (see e.g. Ceri et al., 1990; Eisner, 2023) that iteratively deduces items by applying deduction rules whenever possible to antecedent items that have already been proved; this will eventually deduce all provable items. An unweighted parser extends the recognizer with some extra bookkeeping that lets it return one or more actual proofs of the goal item if it is provable.[6]

4 Earley's Algorithm

Earley's algorithm can be presented as the specific deduction system `Earley` shown in Table 1 (Sikkel, 1997; Shieber et al., 1995; Goodman, 1999), explained in more detail in App. B. Its proof trees $\mathcal{D}_{\text{goal}}$ are in one-to-one correspondence with the derivation trees $\mathcal{T}_{\mathbf{x}}$ (a property that we will maintain for our improved deduction systems in §5 and §7). The grammar $\mathcal{G}$ is encoded by axioms $A \overset{*}{\Rightarrow} \rho$ that correspond to the productions of the grammar. The input sentence $\mathbf{x}$ is encoded by axioms of the form $[k-1, k, a]$ where $a \in \Sigma$; this axiom is true iff $\mathbf{x}_{k-1:k} = x_k = a$.[7] The remaining items have the form $[i, j, A \to \mu \bullet \nu]$, where $0 \le i \le j \le N$, so that the **span** (i, j) refers to a substring $\mathbf{x}_{i:j} \overset{\text{def}}{=} x_{i+1} \cdots x_j$ of the input sentence $\mathbf{x} = x_1 x_2 \ldots x_N$. The item $[i, j, A \to \mu \bullet \nu]$ is derivable only if the grammar $\mathcal{G}$ has a production $A \to \mu\,\nu$ such that $\mu \overset{*}{\Rightarrow} \mathbf{x}_{i:j}$. Therefore, $\bullet$ indicates the progress we have made through the production. An item with nothing to the right of $\bullet$, e.g., $[i, j, A \to \rho \bullet]$, is called **complete**. The set of all items with a shared right index j is called the **item set** of j, denoted $\mathcal{T}_j$.

While $\mu \overset{*}{\Rightarrow} \mathbf{x}_{i:j}$ is a necessary condition for $[i, j, A \to \mu \bullet \nu]$ to be provable, it is not sufficient. For efficiency, the `Earley` deduction system is cleverly constructed so that this item is provable iff[8] it can appear in a proof of the goal item for some input string beginning with $\mathbf{x}_{0:j}$, and thus possibly for $\mathbf{x}$ itself.[9]

Including $[0, 0, S' \to \bullet S]$ as an axiom in the system effectively causes forward chaining to start looking for a derivation at position 0. Forward chaining will prove the goal item $[0, N, S' \to S \bullet]$ iff $S \overset{*}{\Rightarrow} \mathbf{x}$. These two items conveniently pretend that the grammar has been augmented with a new start symbol $S' \notin \mathcal{N}$ that only rewrites according to the single production $S' \to S$.

The `Earley` system employs three deduction rules: PREDICT, SCAN, and COMPLETE. We refer the reader to App. B for a presentation and analysis of these rules, which reveals a total runtime of $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$. App. C outlines how past work improved this runtime. In particular, Graham et al. (1980) presented an unweighted recognizer that is a variant of Earley's, along with implementation details that enable it to run in time $\mathcal{O}\big(N^3|\mathcal{G}|\big)$. However, those details were lost in retelling their algorithm as a deduction system (Sikkel, 1997, p. 113). Our improved deduction system in the next section does enable the $\mathcal{O}\big(N^3|\mathcal{G}|\big)$ runtime, with execution details of forward chaining spelled out in App. H.

[6] Each proved item stores a "backpointer" to the rule that proved it. Equivalently, an item's proofs may be tracked by its weight in a "derivation semiring" (Goodman, 1999).

[7] All methods in this paper can be also applied directly to lattice parsing, in which i, j, k range over states in an acyclic lattice of possible input strings, and 0 and N refer to the unique initial and final states. A lattice edge from j to k labeled with terminal a is then encoded by the axiom $[j, k, a]$.

[8] Assuming that all nonterminals $B \in \mathcal{N}$ are **generating**, i.e., $\exists \mathbf{x}' \in \Sigma^*$ such that $B \overset{*}{\Rightarrow} \mathbf{x}'$. To ensure this, repeatedly mark $B \in \mathcal{N}$ as generating whenever $\mathcal{R}$ contains some $B \to \rho$ such that all nonterminals in ρ are already marked as generating. Then delete any unmarked nonterminals and their rules.

[9] Earley (1970) also generalized the algorithm to prove this item only if it can appear in a proof of some string that begins with $\mathbf{x}_{0:(j+\Delta)}$, for a fixed Δ. This is lookahead of Δ tokens.

5 An Improved Deduction System

Our `EarleyFast` deduction system, shown in the right column of Table 1, shaves a factor of $\mathcal{O}(\mathcal{R})$ from the runtime of `Earley`. It does so by effectively applying a weighted fold transform (Tamaki and Sato, 1984; Eisner and Blatz, 2007; Johnson, 2007) on PRED (§5.1) and COMP (§5.2), introducing coarse-grained items of the forms $[i, j, A \to \bullet \star]$ and $[i, j, A \to \star \bullet]$. In these items, the constant symbol $\star$ can be regarded as a wildcard that stands for "any sequence ρ." We also use these new items to replace the goal item and the axiom that used S'; the extra S' symbol is no longer needed. The proofs are essentially unchanged (App. D).

We now describe our new deduction rules for COMP and PRED. (SCAN is unchanged.) We also analyze their runtime, using the same techniques as in App. B.

5.1 Predict

We split PRED into two rules: PRED1 and PRED2. The first rule, PRED1, creates an item that gathers together all requests to look for a given nonterminal B starting at a given position j:

$$\text{PRED1: } \frac{}{[j, j, B \to \bullet \star]} \; [i, j, A \to \mu \bullet B \, \nu]$$

There are three free choices in the rule: indices i and j, and dotted production $A \to \bullet \, B \, \nu$. Therefore, PRED1 has a total runtime of $\mathcal{O}\big(N^2|\mathcal{G}|\big)$.

The second rule, PRED2, expands the item into commitments to look for each specific kind of B:

$$\text{PRED2: } \frac{B \to \rho}{[j, j, B \to \bullet \rho]} \; [j, j, B \to \bullet \star]$$

PRED2 has two free choices: index j and production $B \to \rho$. Therefore, PRED2 has a runtime of $\mathcal{O}(N|\mathcal{R}|)$, which is dominated by $\mathcal{O}(N|\mathcal{G}|)$ and so the two rules together have a runtime of $\mathcal{O}\big(N^2|\mathcal{G}|\big)$.

5.2 Complete

We speed up COMP in a similar fashion to PRED. We split COMP into two rules: COMP1 and COMP2. The first rule, COMP1, gathers all complete B constituents over a given span into a single item:

$$\text{COMP1: } \frac{[j, k, B \to \rho \bullet]}{[j, k, B \to \star \bullet]}$$

We have three free choices: indices j and k, and complete production $B \to \rho$ with domain size $|\mathcal{R}|$. Therefore, COMP1 has a total runtime of $\mathcal{O}\big(N^2|\mathcal{R}|\big)$, or $\mathcal{O}\big(N^2|\mathcal{G}|\big)$.

The second rule, COMP2, attaches the resulting complete items to any incomplete items that predicted them:

$$\text{COMP2: } \frac{[i, j, A \to \mu \bullet B \, \nu] \quad [j, k, B \to \star \bullet]}{[i, k, A \to \mu \, B \bullet \nu]}$$

We have four free choices: indices i, j, and k, and dotted production $A \to \mu \bullet B \, \nu$. Therefore, COMP2 has a total runtime of $\mathcal{O}\big(N^3|\mathcal{G}|\big)$ and so the two rules together have a runtime of $\mathcal{O}\big(N^3|\mathcal{G}|\big)$.

6 Semiring-Weighted Parsing

We have so far presented Earley's algorithm and our improved deduction system in the unweighted case. However, we are often interested in determining not just whether a parse exists, but the total weight of all parses as in equation (2), or the total weight of all parses consistent with a given prefix as in equation (3).

We first observe that by design, the derivation trees of the CFG are in 1-1 correspondence with the proof trees of our deduction system that are rooted at the goal item. Furthermore, the weight of a derivation subtree can be found as the weight of the corresponding proof tree, if the weight $w(d_V)$ of any proof tree d_V is defined recursively as follows.

Base case: d_V may be a single node, i.e., V is an axiom. If V has the form $A \to \rho$, then $w(d_V)$ is the weight of the corresponding grammar production, i.e., $w(A \to \rho)$. All other axiomatic proof trees of `Earley` and `EarleyFast` have weight ①.[10]

Recursive case: If the root node of d_V has child subtrees $d_{U_1}, d_{U_2}, \ldots$, then $w(d_V) = w(d_{U_1}) \otimes w(d_{U_2}) \otimes \cdots$. However, the factors in this product include only the antecedents written above the bar, not the side conditions (see §3).

Following Goodman (1999), we may also associate a weight with each item V, denoted $\dot{\beta}(V)$, which is the *total* weight of *all* its proofs $d_V \in \mathcal{D}_V$. By the distributive property, we can obtain that weight as an $\oplus$-sum over all one-step proofs of V from antecedents. Specifically, each deduction rule that deduces V contributes an $\oplus$-summand, given

[10] However, this will not be true in `EarleyFSA` (§7 below). There the grammar is given by a WFSA, and each axiom corresponding to an arc or final state of this grammar will inherit its weight from that arc or final state. Similarly, if we generalize to lattice parsing—where the *input* is given by an acyclic WFSA and each proof tree corresponds to a parse of some weighted path from this so-called lattice—then an axiom providing a terminal token should use the weight of the corresponding lattice edge. Then the weight of the proof tree will include the total weight of the lattice path along with the weight of the CFG productions used in the parse.

by the product $\dot{\beta}(U_1) \otimes \dot{\beta}(U_2) \otimes \cdots$ of the weights of its antecedent items (other than side conditions).

Now our weighted recognizer can obtain $Z_{\mathbf{x}}$ (the total weight of all derivations of $\mathbf{x}$) as $\dot{\beta}$ of the goal item (the total weight of all proofs of that item).

For an item V of the form $[i, j, A \to \mu \bullet \nu]$, the weight $\dot{\beta}(V)$ will consider derivations of nonterminals in μ but not those in ν. We therefore refer to $\dot{\beta}(V)$ as an **incomplete inside weight**. However, ν will come into play in the extension of §6.1.

The deduction systems work for any semiring-weighted CFG. Unfortunately, the forward-chaining algorithm for *weighted* deduction (Eisner et al., 2005, Fig. 3) may not terminate if the system permits *cyclic* proofs, where an item can participate in one of its own proofs. In this case, the algorithm will merely approach the correct value of $Z_{\mathbf{x}}$ as it discovers deeper and deeper proofs of the goal item. Cyclicity in our system can arise from sets of unary productions such as $\{A \to B, B \to A\} \subseteq \mathcal{R}$, or equivalently, from $\{A \to E\,B\,E, B \to A\} \subseteq \mathcal{R}$ where $E \overset{*}{\Rightarrow} \varepsilon$ (which is possible if $\mathcal{R}$ contains $E \to \varepsilon$ or other nullary productions). We take the approach of eliminating problematic unary and nullary productions from the weighted grammar without changing $Z_{\mathbf{x}}$ for any $\mathbf{x}$. We provide methods to do this in App. E and App. F respectively. It is important to eliminate nullary productions *before* eliminating unary cycles, since nullary removal may create new unary productions. The elimination of some productions can increase $|\mathcal{G}|$, but we explain how to limit this effect.

6.1 Extension to Prefix Weights

Stolcke (1995) showed how to extend Earley's algorithm to compute prefix probabilities under PCFGs, by associating a "forward probability" with each •-item.[11] However, he relied on the property that all nonterminals A have $Z_A = 1$, where Z_A denotes the **free weight**

$$Z_A \overset{\text{def}}{=} \bigoplus_{T \in \mathcal{T}^A} \bigotimes_{B \to \rho \in T} w(B \to \rho) \qquad (4)$$

As a result, his algorithm does not handle the case of WCFGs or CRF-CFGs (Johnson et al., 1999; Yusuke and Jun'ichi, 2002; Finkel et al., 2008), or even non-tight PCFGs (Chi and Geman, 1998). It also does not handle semiring-weighted grammars. We generalize by associating with each •-item, instead of a "forward probability," a "prefix outside weight" from the same commutative semiring that is used to weight the grammar productions. Formally, each $w(V)$ will now be a pair $(\dot{\beta}(V), \dot{\alpha}(V))$, and we combine these pairs in specific ways.

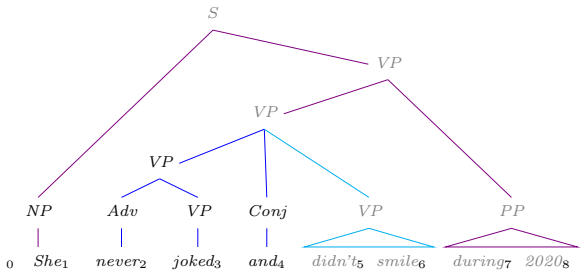

Figure 1: This derivation tree corresponds to a proof under `EarleyFast`. Since $[1, 4, \mathit{VP} \to \mathit{VP}\ \mathit{Conj} \bullet \mathit{VP}]$ is a •-item in that proof, it partitions the steps of that proof into incomplete inside, future inside and outside portions that respectively prove $[1, 4, \mathit{VP} \to \mathit{VP}\ \mathit{Conj} \bullet \mathit{VP}]$, prove the complete item $[1, 6, \mathit{VP} \to \mathit{VP}\ \mathit{Conj}\ \mathit{VP} \bullet]$ from that, and prove goal from the complete item. See Fig. 3 for an alternative derivation and more discussion.

Recall from §4 that the item $V = [i, j, A \to \mu \bullet \nu]$ is provable iff[8] it appears in a proof of some sentence beginning with $\mathbf{x}_{0:j}$. For any such proof containing V, its steps can be partitioned as shown in Fig. 1, factoring the proof weight into three factors. Just as the incomplete inside weight $\dot{\beta}(V)$ is the total weight of all ways to prove V, the **future inside weight** Z_ν is the total weight of all ways to prove $[i, j, A \to \mu\,\nu \bullet]$ from V and the **prefix outside weight** $\dot{\alpha}(V)$ is the total weight of all ways to prove the goal item from $[i, j, A \to \mu\,\nu \bullet]$—in both cases allowing *any* future words $\mathbf{x}_{j:}$ as "free" axioms.[12]

The future inside weight $Z_\nu = \prod_{i:\nu_i \in \mathcal{N}} Z_{\nu_i}$ does not depend on the input sentence. To avoid a slowdown at parsing time, we precompute this product for each suffix ν of each production in $\mathcal{R}$, after using methods in App. F to precompute the free weights Z_A for each nonterminal A.

Like $\dot{\beta}(V)$, $\dot{\alpha}(V)$ is obtained as an $\oplus$-sum over all one-step proofs of V. Typically, each one-step proof increments $\dot{\alpha}(V)$ by the prefix outside weight of its •-antecedent or •-side condition (for COMP2, the *left* •-antecedent). As an important exception, when $V = [j, j, B \to \bullet\star]$, each of its one-step

[11] Also other CFG parsing algorithms can be adapted to compute prefix probabilities, e.g., CKY (Jelinek and Lafferty, 1991; Nowak and Cotterell, 2023).

[12] Prefix outside weights differ from traditional outside weights (Baker, 1979; Lari and Young, 1990; Eisner, 2016), which restrict to the *actual* future words $\mathbf{x}_{j:n}$.

proofs via PRED1 instead increments $\dot{\alpha}(V)$ by

$$\dot{\alpha}([i, j, A \to \mu \bullet B\, \nu]) \\ \otimes \dot{\beta}([i, j, A \to \mu \bullet B\, \nu]) \otimes Z_\nu \quad (5)$$

combining the steps outside $[i, j, A \to \mu \bullet B\, \nu]$ with some steps inside the A (including its production) to get all the steps outside the B. The base case is the start axiom, $\dot{\alpha}([0, 0, S \to \bullet \star]) = ①$.

Unfortunately, this computation of $\dot{\alpha}(V)$ is only correct if there is no left-recursion in the grammar. We explain this issue in App. G.1 and fix it by extending the solution of Stolcke (1995, §4.5.1).

The prefix weight of $\mathbf{x}_{0:j}$ $(j > 0)$ is computed as an $\oplus$-sum $\dot{\alpha}([j, j])$ over all one-step proofs of the new item $[j, j]$ via the following new deduction rule that is triggered by the consequent of SCAN:

$$\text{POS: } \frac{}{[j, j]}\ [i, j, A \to \mu\, a \bullet \nu]$$

Each such proof increments the prefix weight by

$$\dot{\alpha}([i, j, A \to \mu\, a \bullet \nu]) \\ \otimes \dot{\beta}([i, j, A \to \mu\, a \bullet \nu]) \otimes Z_\nu \quad (6)$$

7 Earley's Algorithm Using an FSA

In this section, we present a generalization of `EarleyFast` that can parse with any **weighted finite-state automaton** (**WFSA**) grammar $\mathcal{M}$ in $\mathcal{O}\big(N^3|\mathcal{M}|\big)$. Here $\mathcal{M}$ is a WFSA (Mohri, 2009) that encodes the CFG productions as follows. For any $\rho \in (\Sigma \cup \mathcal{N})^*$ and any $A \in \mathcal{N}$, for $\mathcal{M}$ to accept the string $\rho\, \widehat{A}$ with weight $w \in \mathbb{W}$ is tantamount to having the production $A \to \rho$ in the CFG with weight w. The grammar size $|\mathcal{M}|$ is the number of WFSA arcs. See Fig. 2 for an example.

This presentation has three advantages over a CFG. First, $\mathcal{M}$ can be compiled from an extended CFG (Purdom and Brown, 1981), which allows user-friendly specifications like $NP \to Det?\ Adj^*\ N^+\ PP^*$ that may specify infinitely many productions with unboundedly long right-hand-sides ρ (although $\mathcal{M}$ still only describes a context-free language). Second, productions with similar right-hand-sides can be partially merged to achieve a smaller grammar and a faster runtime. They may share partial paths in $\mathcal{M}$, which means that a single item can efficiently represent many dotted productions. Third, when $\otimes$ is non-commutative, only the WFSA grammar formalism allows elimination of nullary rules in all cases (see App. F).

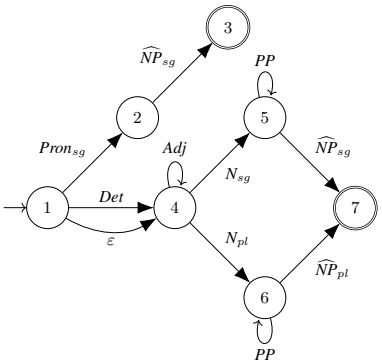

Figure 2: Part of an FSA for English, showing ways to generate singular (sg) and plural (pl) noun phrases. The paths shown here correspond to the extended CFG production rules $NP_{sg} \to (\ Det?\ Adj^*\ N_{sg}^+\ PP^*\ ) \mid Pron_{sg}$ and $NP_{pl} \to Det?\ Adj^*\ N_{pl}^+\ PP^*$.

Our WFSA grammar is similar to a **recursive transition network** or **RTN** grammar (Woods, 1970). Adapting Earley's algorithm to RTNs was discussed by Kochut (1983), Leermakers (1989), and Perlin (1991). Klein and Manning (2001b) used a weighted version for PTB parsing. None of them spelled out a deduction system, however.

Also, an RTN is a collection of productions of the form $A \to \mathcal{M}_A$, where for $\mathcal{M}_A$ to accept ρ corresponds to having $A \to \rho$ in the CFG. Thus an RTN uses one FSA per nonterminal. Our innovation is to use one WFSA for the entire grammar, specifying the left-hand-side nonterminal as a final symbol. Thus, to allow productions $A \to \mu\, \nu$ and $B \to \mu\, \nu'$, our single WFSA can have paths $\mu\, \nu\, \widehat{A}$ and $\mu\, \nu'\, \widehat{B}$ that share the μ prefix—as in Fig. 2. This allows our `EarleyFSA` to match the μ prefix only once, in a way that could eventually result in completing either an A or a B (or both).[13]

A traditional weighted CFG $\mathcal{G}$ can be easily encoded as an acyclic WFSA $\mathcal{M}$ with $|\mathcal{M}| = |\mathcal{G}|$, by creating a weighted path of length k and weight w[14] for each CFG production of size k and weight w, terminating in a final state, and then merging the initial states of these paths into a single state that becomes the initial state of the resulting WFSA. The paths are otherwise disjoint. Importantly, this WFSA can then be determinized and minimized

[13] Nederhof (1994b) also shares prefixes between A and B; but there, once paths split to yield separate items, they cannot remerge to share a *suffix*. We can merge by deriving $[j, k, q?]$ in multiple ways. Our $[j, k, q?]$ does not specify its set of target left-hand sides; FILTER recomputes that set dynamically.

[14] For example, the production $S \to NP\ VP$ would be encoded as a path of length 3 accepting the sequence $NP\ VP\ \widehat{S}$. The production's weight may arbitrarily be placed on the first arc of the path, the other arcs having weight ① (see App. A).

Domains $i, j, k \in \{0, \ldots, N\} \quad A \in \mathcal{N} \quad a \in \Sigma \quad q, q' \in \mathcal{Q}$

Items $[i, j, q] \; [i, j, q?] \; [i, j, a] \; [i, j, A \to \bullet \star] \; [i, j, A \to \star \bullet] \; \underbrace{q \in \mathcal{I} \quad q' \in \mathcal{F} \quad q \overset{a}{\rightsquigarrow} q' \quad q \overset{A}{\rightsquigarrow} q' \quad q \overset{A}{\rightsquigarrow} \star \quad q \overset{*\widehat{A}}{\rightsquigarrow} \star}_{\text{WFSA items}}$

Axioms $[k-1, k, x_k], \forall k \in \{1, \ldots, N\} \qquad [0, 0, S \to \bullet \star]$ WFSA items derived from the WFSA grammar (see §7)

Goals $[0, N, S \to \star \bullet]$

Rules

PRED1: $\dfrac{[i, j, q]}{[j, j, B \to \bullet \star] \quad q \overset{B}{\rightsquigarrow} \star}$

PRED2: $\dfrac{q \in \mathcal{I}}{[j, j, q?]}$

SCAN: $\dfrac{[i, j, q] \quad q \overset{a}{\rightsquigarrow} q' \quad [j, k, a]}{[i, k, q'?]}$

COMP1: $\dfrac{[j, k, q] \quad q \overset{\widehat{B}}{\rightsquigarrow} q' \quad q' \in \mathcal{F}}{[j, k, B \to \star \bullet]}$

COMP2: $\dfrac{[i, j, q] \quad q \overset{B}{\rightsquigarrow} q' \quad [j, k, B \to \star \bullet]}{[i, k, q'?]}$

EPSILON: $\dfrac{[i, j, q] \quad q \overset{\varepsilon}{\rightsquigarrow} q'}{[i, j, q'?]}$

FILTER: $\dfrac{[j, k, q?]}{[j, k, q]} \; \begin{array}{l}[j, j, A \to \bullet \star] \\ q \overset{*\widehat{A}}{\rightsquigarrow} \star\end{array}$

Table 2: `EarleyFSA`, a variant of `EarleyFast` in which FSA states replace dotted productions. Side conditions are stacked horizontally in the interest of space. A faster binarized version is given in supplementary material as Table 6.

(Mohri, 1997) to potentially reduce the number of states and arcs (while preserving the total weight of each sequence) and thus speed up parsing (Klein and Manning, 2001b). Among other things, this will merge common prefixes and common suffixes.

In general, however, the grammar can be specified by any WFSA $\mathcal{M}$—not necessarily deterministic. This could be compiled from weighted regular expressions, or be an encoded Markov model trained on observed productions (Collins, 1999), or be obtained by merging states of another WFSA grammar (Stolcke and Omohundro, 1994) in order to smooth its weights and speed it up.

The WFSA has states $\mathcal{Q}$ and weighted arcs (or edges) $\mathcal{E}$, over an alphabet $\mathcal{A}$ consisting of $\Sigma \cup \mathcal{N}$ together with hatted nonterminals like $\widehat{A}$. Its initial and final states are denoted by $\mathcal{I} \subseteq \mathcal{Q}$ and $\mathcal{F} \subseteq \mathcal{Q}$, respectively.[15] We denote an arc of the WFSA by $(q \overset{a}{\rightsquigarrow} q') \in \mathcal{E}$ where $q, q' \in \mathcal{Q}$ and $a \in \mathcal{A} \cup \{\varepsilon\}$. This corresponds to an axiom with the same weight as the edge. $q \in \mathcal{I}$ corresponds to an axiom whose weight is the initial-state weight of q. The item $q \in \mathcal{F}$ is true not only if q is a final state but more generally if q has an ε-path of length ≥ 0 to a final state; the item's weight is the total weight of all such ε-paths, where a path's weight includes its final-state weight.

For a state $q \in \mathcal{Q}$ and symbol $A \in \mathcal{N}$, the precomputed side condition $q \overset{A}{\rightsquigarrow} \star$ is true iff there exists a state $q' \in \mathcal{Q}$ such that $q \overset{A}{\rightsquigarrow} q'$ exists in $\mathcal{E}$. Additionally, the precomputed side condition $q \overset{*\widehat{A}}{\rightsquigarrow} \star$ is true if there exists a path starting from q that eventually reads $\widehat{A}$. As these are only used as side conditions, they may be given any non-⓪ weight.

The `EarleyFSA` deduction system is given in Table 2. It can be run in time $\mathcal{O}\big(N^3|\mathcal{M}|\big)$. It is similar to `EarleyFast`, where the dotted rules have been replaced by WFSA states. However, unlike a dotted rule, a state does not specify a PREDICTed left-hand-side nonterminal. As a result, when any deduction rule "advances the dot" to a new state q, it builds a provisional item $[j, k, q?]$ that is annotated with a question mark. This mark represents the fact that although q is compatible with several left hand sides A (those for which $q \overset{*\widehat{A}}{\rightsquigarrow} \star$ is true), the left context $\mathbf{x}_{0:j}$ might not call for any of those nonterminals. If it calls for at least one such nonterminal A, then the new FILTER rule will remove the question mark, allowing further progress.

One important practical advantage of this scheme for natural language parsing is that it prevents a large-vocabulary slowdown.[16] In `Earley`, applying PREDICT to (say) $[3, 4, \mathit{NP} \to \mathit{Det} \bullet N]$ results in thousands of items of the form $[4, 4, N \to \bullet\, a]$ where a ranges over all nouns in the vocabulary. But `EarleyFSA` in the corresponding situation will predict only $[4, 4, q]$ where q is the initial state, without yet predicting the next word. If the next input word is $[4, 5, \mathit{happy}]$, then `EarleyFSA` follows just the *happy* arcs from q, yielding items of the form $[4, 5, q'?]$ (which will

[15] Note that if the WFSA is obtained as described above, it will only have one initial state.

[16] Earley (1970) used 1-word lookahead for this; see App. G.2.

then be FILTERed away since *happy* is not a noun).

Note that SCAN, COMP1 and COMP2 are ternary, rather than binary as in `EarleyFast`. For further speed-ups we can apply the fold transform on these rules in a similar manner as before, resulting in binary deduction rules. We present this binarized version in App. I.

As before, we must eliminate unary and nullary rules before parsing; App. J explains how to do this with a WFSA grammar. In addition, although Table 2 allows the WFSA to contain ε-arcs, App. J explains how to eliminate *ε-cycles* in the WFSA, which could prevent us from converging, for the usual reason that an item $[i, j, q]$ could participate in its own derivation. Afterwards, there is again a nearly acyclic order in which the deduction engine can prove items (as in App. H.1 or App. H.3).

As noted above, we can speed up `EarleyFSA` by reducing the size of the WFSA. Unfortunately, minimization of general FSAs is NP-hard. However, we can at least seek the minimal *deterministic* WFSA $\mathcal{M}'$ such that $|\mathcal{M}'| \leq |\mathcal{M}|$, at least in most semirings (Mohri, 2000; Eisner, 2003). The determinization (Aho et al., 1986) and minimization (Aho and Hopcroft, 1974; Revuz, 1992) algorithms for the boolean semiring are particularly well-known. Minimization merges states, which results in merging items, much as when `EarleyFast` merged items that had different pre-dot symbols (Leermakers, 1992; Nederhof and Satta, 1997; Moore, 2000).

Another advantage of the WFSA presentation of Earley's is that it makes it simple to express a tighter bound on the runtime. Much of the grammar size $|\mathcal{G}|$ or $|\mathcal{M}|$ is due to terminal symbols that are not used at most positions of the input. Suppose the input is an ordinary sentence (one word at each position, unlike the lattice case in footnote 7), and suppose c is a constant such that no state q has more than c outgoing arcs labeled with the same terminal $a \in \Sigma$. Then when SCAN tries to extend $[i, j, q]$, it considers at most c arcs. Thus, the $\mathcal{O}(|\mathcal{M}|)$ factor in our runtime (where $|\mathcal{M}| = |\mathcal{E}|$) can be replaced with $\mathcal{O}(|\mathcal{Q}| \cdot c + |\mathcal{E}_{\mathcal{N}}|)$, where $\mathcal{E}_{\mathcal{N}} \subseteq \mathcal{E}$ is the set of edges that are *not* labeled with terminals.

8 Practical Runtime of Earley's

We empirically measure the runtimes of `Earley`, `EarleyFast`, and `EarleyFSA`. We use the tropical semiring to find the highest-weighted derivation trees. We use two grammars that were extracted from the PTB: Markov-order-2 (M2) and Parent-annotated Markov-order-2 (PM2).[17] For each grammar, we ran our parsers (using the tropical semiring; Pin, 1998) on 100 randomly selected sentences of 5 to 40 words from the PTB test-set (mean 21.4, stdev 10.7), although we omitted sentences of length > 25 from the `Earley` graph as it was too slow (> 3 minutes per sentence). The full results are displayed in App. L. The graph shows that `EarleyFast` is roughly $20\times$ faster at all sentence lengths. We obtain a further speed-up of $2.5\times$ by switching to `EarleyFSA`.

9 Conclusion

In this reference work, we have shown how the runtime of Earley's algorithm is reduced to $\mathcal{O}\big(N^3|\mathcal{G}|\big)$ from the naive $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$. We presented this dynamic programming algorithm as a deduction system, which splits prediction and completion into two steps each, in order to share work among related items. To further share work, we generalized Earley's algorithm to work with a grammar specified by a weighted FSA. We demonstrated that these speed-ups are effective in practice. We also provided details for efficient implementation of our deduction system. We showed how to generalize these methods to semiring-weighted grammars by correctly transforming the grammars to eliminate cyclic derivations. We further provided a method to compute the total weight of all sentences with a given prefix under a semiring-weighted CFG.

We intend this work to serve as a clean reference for those who wish to efficiently implement an Earley-style parser or develop related incremental parsing methods. For example, our deduction systems could be used as the starting point for

- neural models of incremental processing, in which each derivation of an item contributes not only to its weight but also to its representation in a vector space (cf. Drozdov et al., 2019; Mei et al., 2020);
- biasing an autoregressive language model toward high-weighted grammatical prefixes via product-of-experts decoding (cf. Shin et al., 2021; Roy et al., 2022; Fang et al., 2023);
- extensions to incremental parsing of more or less powerful grammar formalisms.

[17] Available at `https://code.google.com/archive/p/bubs-parser/`. M2 contains 52,009 preterminal rules and 13,893 other rules. PM2 contains 52,009 preterminal rules and 25,919 other rules. The downloaded grammars did not have nullary rules or unary chains.

10 Limitations

Orthogonal to the speed-ups discussed in this work, Earley (1970) described an extension that we do not include here, which filters deduction items using k words of lookahead. (However, we do treat 1-word lookahead and left-corner parsing in App. G.2.)

While our deduction system runs in time proportional to the grammar size $|\mathcal{G}|$, this size is measured only after unary and nullary productions have been eliminated from the grammar—which can increase the grammar size as discussed in Apps. E and F.

We described how to compute prefix weights only for `EarleyFast`, and we gave a prioritized execution scheme (App. H.3) only for `EarleyFast`. The versions for `EarleyFSA` should be similar.

Computing sentence weights (2) and prefix weights (3) involves a sum over infinitely many trees. In arbitrary semirings, there is no guarantee that such sums can be computed. Computing them requires summing geometric series and—more generally—finding minimal solutions to systems of polynomial equations. See discussion in App. A and App. F. Non-commutative semirings also present special challenges; see App. K.

Acknowledgments

We thank Mark-Jan Nederhof for useful references and criticisms, and several anonymous reviewers for their feedback. Any remaining errors are our own.

Andreas Opedal is supported by the Max Planck ETH Center for Learning Systems.

References

Alfred V. Aho and John E. Hopcroft. 1974. *The Design and Analysis of Computer Algorithms*. Pearson Education.

Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. 1986. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley series in computer science / World student series edition. Addison-Wesley.

John Aycock and R. Nigel Horspool. 2002. Practical Earley parsing. *The Computer Journal*, 45(6):620–630.

J. K. Baker. 1979. Trainable grammars for speech recognition. In *Speech Communication Papers Presented at the 97th Meeting of the Acoustical Society of America*, MIT, Cambridge, MA.

François Barthélemy. 1993. *Outils pour lÁnalyse Syntaxique Contextuelle*. Ph.D. thesis, University of Orléans.

Stefano Ceri, Georg Gottlob, and Letizia Tanca. 1990. *Logic Programming and Databases*. Surveys in computer science. Springer.

Zhiyi Chi and Stuart Geman. 1998. Estimation of probabilistic context-free grammars. *Computational Linguistics*, 24(2):299–305.

Michael J. Collins. 1999. *Head-Driven Statistical Models for Natural Language Parsing*. Ph.D. thesis, University of Pennsylvania.

Eric V. de la Clergerie. 1993. *Automates a piles et programmation dynamique DyAlog: une application a la programmation en logique*. Ph.D. thesis, University Paris VII.

Robert B. Dial. 1969. Algorithm 360: Shortest-path forest with topological ordering. *Communications of the ACM*, 12(11):632–633.

Andrew Drozdov, Patrick Verga, Mohit Yadav, Mohit Iyyer, and Andrew McCallum. 2019. Unsupervised latent tree induction with deep inside-outside recursive auto-encoders. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1129–1141, Minneapolis, Minnesota. Association for Computational Linguistics.

Aaron Dunlop, Nathan Bodenstab, and Brian Roark. 2010. Reducing the grammar constant: an analysis of CYK parsing efficiency. Technical report, Oregon Health & Science University.

Jay Earley. 1968. *An Efficient Context-Free Parsing Algorithm*. Ph.D. thesis, Carnegie-Mellon University.

Jay Earley. 1970. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102.

Jason Eisner. 2003. Simpler and more general minimization for weighted finite-state automata. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 64–71.

Jason Eisner. 2016. Inside-outside and forward-backward algorithms are just backprop (tutorial paper). In *Proceedings of the Workshop on Structured Prediction for NLP@EMNLP 2016, Austin, TX, USA, November 5, 2016*.

Jason Eisner. 2023. Time-and-space-efficient weighted deduction. *Transactions of the Association for Computational Linguistics*. Accepted for publication.

Jason Eisner and John Blatz. 2007. Program transformations for optimization of parsing algorithms and other weighted logic programs. In *Proceedings of FG 2006: The 11th Conference on Formal Grammar*, pages 45–85. CSLI Publications.

Jason Eisner, Eric Goldlust, and Noah A. Smith. 2005. Compiling comp ling: Weighted dynamic programming and the Dyna language. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 281–290, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Javier Esparza, Stefan Kiefer, and Michael Luttenberger. 2007. An extension of Newton's method to ω-continuous semirings. In *Proceedings of the International Conference on Developments in Language Theory*, volume 4588 of *Lecture Notes in Computer Science*, pages 157–168. Springer.

Kousha Etessami and Mihalis Yannakakis. 2009. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the Association for Computing Machinery*, 56(1).

Hao Fang, Anusha Balakrishnan, Harsh Jhamtani, John Bufe, Jean Crawford, Jayant Krishnamurthy, Adam Pauls, Jason Eisner, Jacob Andreas, and Dan Klein. 2023. The whole truth and nothing but the truth: Faithful and controllable dialogue response generation with dataflow transduction and constrained decoding. In *Findings of the Association for Computational Linguistics (ACL)*, Toronto, Canada.

Jenny Rose Finkel, Alex Kleeman, and Christopher D. Manning. 2008. Efficient, feature-based, conditional random field parsing. In *Proceedings of ACL-08: HLT*, pages 959–967.

Joshua Goodman. 1999. Semiring parsing. *Computational Linguistics*, 25(4):573–606.

Susan L. Graham, Michael A. Harrison, and Walter L. Ruzzo. 1980. An improved context-free recognizer. *ACM Transactions on Programming Languages and Systems*, 2(3):415–462.

John Hale. 2001. A probabilistic Earley parser as a psycholinguistic model. In *Second Meeting of the North American Chapter of the Association for Computational Linguistics*.

Lisa Higham and Eric Schenk. 1993. PRAM memory allocation and initialization. *Parallel Processing Letters*, 3(3):291–299.

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2007. *Introduction to Automata Theory, Language, and Computation*, 3 edition. Pearson international edition. Addison-Wesley.

Frederick Jelinek and John D. Lafferty. 1991. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–353.

Mark Johnson. 2000. Inside-outside (computer program).

Mark Johnson. 2007. Transforming projective bilexical dependency grammars into efficiently-parsable CFGs with unfold-fold. In *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics*, pages 168–175, Prague, Czech Republic. Association for Computational Linguistics.

Mark Johnson, Stuart Geman, Stephen Canon, Zhiyi Chi, and Stefan Riezler. 1999. Estimators for stochastic "unification-based" grammars. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 535–541, College Park, Maryland, USA. Association for Computational Linguistics.

Mark Johnson and Brian Roark. 2000. Compact non-left-recursive grammars using the selective left-corner transform and factoring. In *COLING 2000 Volume 1: The 18th International Conference on Computational Linguistics*.

Daniel Jurafsky and James H. Martin. 2009. *Speech and Language Processing*, 2 edition. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Arthur B. Kahn. 1962. Topological sorting of large networks. *Commmunications of the ACM*, 5(11):558–562.

Dan Klein and Christopher D. Manning. 2001a. Parsing and hypergraphs. In *Proceedings of the Seventh International Workshop on Parsing Technologies*, pages 123–134, Beijing, China.

Dan Klein and Christopher D. Manning. 2001b. Parsing with treebank grammars: Empirical bounds, theoretical models, and the structure of the Penn Treebank. In *Proceedings of the 39th Annual Meeting of the Association for Computational Linguistics*, pages 338–345, Toulouse, France. Association for Computational Linguistics.

Donald Ervin Knuth. 1997. *The art of computer programming, Volume I: Fundamental Algorithms*, 3 edition. Addison-Wesley.

Krzysztof Kochut. 1983. Towards the elastic ATN implementation. In *The Design of Interpreters, Compilers, and Editors for Augmented Transition Networks*, pages 175–214. Springer.

Werner Kuich. 1997. Semirings and formal power series: Their relevance to formal languages and automata. In *Handbook of Formal Languages: Word, Language, Grammar*, volume 1, pages 609–677. Springer.

K. Lari and S.J. Young. 1990. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4(1):35–56.

René Leermakers. 1989. How to cover a grammar. In *27th Annual Meeting of the Association for Computational Linguistics*, pages 135–142, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

René Leermakers. 1992. A recursive ascent Earley parser. *Information Processing Letters*, 41(2):87–91.

Daniel J. Lehmann. 1977. Algebraic structures for transitive closure. *Theoretical Computer Science*, 4(1):59–76.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn treebank. *Computational Linguistics*, 19(2):313–330.

David A. McAllester. 2002. On the complexity analysis of static analyses. *Journal of the ACM*, 49(4):512–537.

Hongyuan Mei, Guanghui Qin, Minjie Xu, and Jason Eisner. 2020. Neural Datalog through time: Informed temporal modeling via logical specification. In *Proceedings of the 37th International Conference on Machine Learning*.

Guido Minnen. 1996. Magic for filter optimization in dynamic bottom-up processing. In *Proceedings of the 34th conference on Association for Computational Linguistics*, pages 247–254.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

Mehryar Mohri. 2000. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 324:177–201.

Mehryar Mohri. 2002. Generic ε-removal and input ε-normalization algorithms for weighted transducers. *International Journal of Foundations of Computer Science*, 13(1):129–143.

Mehryar Mohri. 2009. Weighted automata algorithms. In *Handbook of Weighted Automata*, chapter 6. Springer, Berlin, Heidelberg.

Robert C. Moore. 2000. Improved left-corner chart parsing for large context-free grammars. In *Proceedings of the Sixth International Workshop on Parsing Technologies*, pages 171–182, Trento, Italy. Association for Computational Linguistics.

Mark J. Nederhof. 1994a. *Linguistic Parsing and Program Transformations*. Ph.D. thesis, University of Nijmegen.

Mark-Jan Nederhof. 1993. Generalized left-corner parsing. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics*, Utrecht, The Netherlands. Association for Computational Linguistics.

Mark-Jan Nederhof. 1994b. An optimal tabular parsing algorithm. In *32nd Annual Meeting of the Association for Computational Linguistics*, pages 117–124, Las Cruces, New Mexico, USA. Association for Computational Linguistics.

Mark-Jan Nederhof and G. Satta. 1997. A variant of earley parsing. In *International Conference of the Italian Association for Artificial Intelligence*.

Mark-Jan Nederhof and Giorgio Satta. 2008. Computing partition functions of PCFGs. *Research on Language and Computation*, 6(2):139–162.

Franz Nowak and Ryan Cotterell. 2023. A faster algorithm for computing prefix probabilities. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL)*, Toronto, Canada.

Fernando C. N. Pereira and Stuart M. Shieber. 1987. *Prolog and Natural-Language Analysis*. Number 10 in CSLI Lecture Notes. Center for the Study of Language and Information.

Fernando C. N. Pereira and David H. D. Warren. 1983. Parsing as deduction. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 137–144, Cambridge, Massachusetts, USA. Association for Computational Linguistics.

Mark Perlin. 1991. LR recursive transition networks for Earley and Tomita parsing. In *29th Annual Meeting of the Association for Computational Linguistics*, pages 98–105, Berkeley, California, USA. Association for Computational Linguistics.

Slav Petrov, Leon Barrett, Romain Thibaux, and Dan Klein. 2006. Learning accurate, compact, and interpretable tree annotation. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 433–440, Sydney, Australia. Association for Computational Linguistics.

Benjamin C. Pierce. 2002. *Types and Programming Languages*. MIT Press.

Jean-Eric Pin. 1998. Tropical Semirings. In J. Gunawardena, editor, *Idempotency (Bristol, 1994)*, Publ. Newton Inst. 11, pages 50–69. Cambridge Univ. Press, Cambridge.

Sinan Polat, Merve Selcuk-Simsek, and Ilyas Cicekli. 2016. A modified earley parser for huge natural language grammars. *Res. Comput. Sci.*, 117:23–35.

Paul Walton Purdom, Jr. and Cynthia A. Brown. 1981. Parsing extended *LR(k)* grammars. *Acta Informatica*, 15:115–127.

Raghu Ramakrishnan. 1991. Magic templates: A spellbinding approach to logic programs. *Journal of Logic Programming*, 11(3-4):189–216.

Dominique Revuz. 1992. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1):181–189.

Brian Roark. 2001. Probabilistic top-down parsing and language modeling. *Computational Linguistics*, 27(2):249–276.

Daniel J. Rosenkrantz and Philip M. Lewis. 1970. Deterministic left corner parsing. In *11th Annual Symposium on Switching and Automata Theory*. IEEE.

Subhro Roy, Sam Thomson, Tongfei Chen, Richard Shin, Adam Pauls, Jason Eisner, and Benjamin Van Durme. 2022. Benchclamp: A benchmark for evaluating language models on semantic parsing.

Stuart M. Shieber, Yves Schabes, and Fernando C.N. Pereira. 1995. Principles and implementation of deductive parsing. *The Journal of Logic Programming*, 24(1):3–36.

Richard Shin, Christopher H. Lin, Sam Thomson, Charles Chen, Subhro Roy, Emmanouil Antonios Platanios, Adam Pauls, Dan Klein, Jason Eisner, and Benjamin Van Durme. 2021. Constrained language models yield few-shot semantic parsers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Punta Cana.

Klaas Sikkel. 1997. *Parsing Schemata - A Framework for Specification and Analysis of Parsing Algorithms*. Texts in Theoretical Computer Science. An EATCS Series. Springer.

Andreas Stolcke. 1995. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):165–201.

Andreas Stolcke and Stephen M. Omohundro. 1994. Best-first model merging for hidden Markov model induction. Technical Report ICSI TR-94-003, ICSI, Berkeley, CA.

Hisao Tamaki and Taisuke Sato. 1984. Unfold/fold transformation of logic programs. In *Proceedings of the Second International Logic Programming Conference, Uppsala University, Uppsala, Sweden, July 2-6, 1984*, pages 127–138.

Robert E. Tarjan. 1972. Depth-first search and linear graph algorithms. *SIAM J. Computing*, 1(2):146–160.

Robert Endre Tarjan. 1981a. Fast algorithms for solving path problems. *Journal of the ACM*, 28(3):594–614.

Robert Endre Tarjan. 1981b. A unified approach to path problems. *Journal of the ACM*, 28(3):577–593.

Mikkel Thorup. 2000. On RAM priority queues. *SIAM Journal on Computing*, 30(1):86–109.

William A. Woods. 1970. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606.

Miyao Yusuke and Tsujii Jun'ichi. 2002. Maximum entropy estimation for feature forests. In *Proceedings of the Second International Conference on Human Language Technology Research*, HLT '02, page 292–297, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

A Semirings

As mentioned in §2, the definition of weighted context-free grammars rests on the definition of semirings. A semiring $\mathcal{S}$ is a 5-tuple $\langle \mathbb{W}, \oplus, \otimes, \mathbb{0}, \mathbb{1} \rangle$, where the set $\mathbb{W}$ is equipped with two operators: $\oplus$, which is associative and commutative, and $\otimes$, which is associative and distributes over $\oplus$. The semiring contains values $\mathbb{0}, \mathbb{1} \in \mathbb{W}$ such that $\mathbb{0}$ is an identity element for $\oplus$ ($w \oplus \mathbb{0} = \mathbb{0} \oplus w = w, \forall w \in \mathbb{W}$) and annihilator for $\otimes$ ($w \otimes \mathbb{0} = \mathbb{0} \otimes w = \mathbb{0}, \forall w \in \mathbb{W}$) and $\mathbb{1}$ is an identity for $\otimes$ ($w \otimes \mathbb{1} = \mathbb{1} \otimes w = w, \forall w \in \mathbb{W}$).

A semiring is commutative if additionally $\otimes$ is commutative. A closed semiring has an additional operator $*$ satisfying the axiom $(\forall w \in \mathbb{W})\ w^* = \mathbb{1} \oplus w \otimes w^* = \mathbb{1} \oplus w^* \otimes w$. The interpretation is that w^* returns the infinite sum $\mathbb{1} \oplus w \oplus (w \otimes w) \oplus (w \otimes w \otimes w) \oplus \cdots$.

As an example that may be of particular interest, Goodman (1999) shows how to construct a (non-commutative) derivation semring, so that $\mathrm{Z}_{\mathbf{x}}$ in equation (2) gives the best derivation (parse tree) along with its weight, or alternatively a representation of the forest of all weighted derivations. This is how a weighted recognizer can be converted to a parser.

B Earley's Original Algorithm as a Deduction System

§4 introduced the deduction system that corresponds to Earley's original algorithm. We explain and analyze it here. Overall, the three rules of this system, `Earley` (Table 1), correspond to possible steps in a top-down recursive descent parser (Aho et al., 1986):

- SCAN consumes the next single input symbol (the base case of recursive descent);
- PREDICT calls a subroutine to consume an entire constituent of a given nonterminal type by recursively consuming its subconstituents;
- COMPLETE returns from that subroutine.

How then does it differ from recursive descent? Rather like depth-first search, Earley's algorithm uses memoization to avoid redoing work, which avoids exponential-time backtracking and infinite recursion. But like breadth-first search, it pursues possibilities in parallel rather than by backtracking. The steps are invoked not by a backtracking call stack but by a deduction engine, which can deduce

new items in any convenient order. The effect on the recursive descent parser is essentially to allow co-routining (Knuth, 1997): execution of a recursive descent subroutine can suspend until further input becomes available or until an ancestor routine has returned and memoized a result thanks to some other nondeterministic execution path.

B.1 Predict

To look for constituents of type B starting at position j, using the rule $B \to \rho$, we need to prove $[j, j, B \to \bullet\, \rho]$. Earley's algorithm imposes $[i, j, A \to \mu \bullet B\, \nu]$ as a side condition, so that we only start looking if such a constituent B could be combined with some item to its left.[18]

$$\textsc{Pred}: \frac{B \to \rho}{[j, j, B \to \bullet\, \rho]}\ [i, j, A \to \mu \bullet B\, \nu]$$

Runtime analysis. How many ways are there to jointly instantiate the two antecedents of PRED with actual items? The pair of items is determined by making four choices:[19] indices i and j with a domain size of $N+1$, dotted production $A \to \mu \bullet B\, \nu$ with domain size $|\mathcal{G}|$, and production $B \to \rho$ with a domain size of $|\mathcal{R}|$. Therefore, the number of instantiations of PRED is $\mathcal{O}\big(N^2|\mathcal{G}||\mathcal{R}|\big)$. That is then PRED's contribution to the runtime of a suitable implementation of forward chaining deduction, using Theorem 1 of McAllester (2002).[20]

B.2 Scan

If we have proved an incomplete item $[i, j, A \to \mu \bullet a\, \nu]$, we can advance the dot if the next terminal symbol is a:

$$\textsc{Scan}: \frac{[i, j, A \to \mu \bullet a\, \nu] \quad [j, k, a]}{[i, k, A \to \mu\, a \bullet \nu]}$$

[18] Minnen (1996) and Eisner and Blatz (2007) explain that this side condition is an instance of the "magic sets" technique that filters some unnecessary work from a bottom-up algorithm (Ramakrishnan, 1991).

[19] Treating these choices as free and independent is enough to give us an upper bound. In actuality, the choices are not quite independent—for example, any provable item has $i \leq j$—but there are no interdependencies that could be exploited to tighten our asymptotic bound.

[20] Technically, that theorem also directs us to count the instantiations of just the first antecedent, namely $\mathcal{O}(|\mathcal{R}|)$. But this term can be ignored, as it is dominated in the asymptotic analysis by the number of complete instantiations $\mathcal{O}\big(N^2|\mathcal{G}||\mathcal{R}|\big)$. In general, we can stick to upper-bounding the number of complete instantiations whenever this upper bound treats the choices as independent, since then it always equals or exceeds the number of partial instantiations.

This makes progress toward completing the A. Note that SCAN pushes the antecedent to a subsequent item set $\mathcal{T}_k$. Since terminal symbols have a span width of 1, it follows that $j = k - 1$.

Runtime analysis. SCAN has three free choices: indices i and j with a domain size of $N + 1$, and dotted production $A \to \mu \bullet B\, \nu$ with domain size $|\mathcal{G}|$. Therefore, SCAN contributes $\mathcal{O}\big(N^2|\mathcal{G}|\big)$ to the overall runtime.

B.3 Complete

Recall that having $[i, j, A \to \mu \bullet B\, \nu]$ allowed us to start looking for a B at position j (PRED). Once we have found a complete B by deriving $[j, k, B \to \rho \bullet]$, we can advance the dot in the former rule:

$$\textsc{Comp}: \frac{[i, j, A \to \mu \bullet B\, \nu] \quad [j, k, B \to \rho \bullet]}{[i, k, A \to \mu\, B \bullet \nu]}$$

Runtime analysis. COMP has five free choices: indices i, j, and k with a domain size of $N + 1$, dotted production $A \to \mu \bullet B\, \nu$ with domain size $|\mathcal{G}|$, and the complete production $B \to \rho$ with a domain size of $|\mathcal{R}|$. Therefore, COMP contributes $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$ to the runtime.

B.4 Total Space and Runtime

By a similar analysis of free choices, the number of items that the `Earley` deduction system will be able to prove is $\mathcal{O}\big(N^2|\mathcal{G}|\big)$. This is a bound on the space needed by the forward chaining implementation to store the items that have been proved so far and index them for fast lookup (McAllester, 2002; Eisner et al., 2005; Eisner, 2023).

Following Theorem 1 of McAllester (2002), adding this count to the total number of rule instantiations from the above sections yields a bound on the total runtime of the `Earley` algorithm, namely $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$ as claimed.

C Previous Speed-ups

We briefly discuss past approaches used to improve the asymptotic efficiency of `Earley`.

Leermakers (1992) noted that in an item of the form $[i, j, A \to \mu \bullet \nu]$, the sequence μ is irrelevant to subsequent deductions. Therefore, he suggested (in effect) replacing μ with a generic placeholder $\star$. This merges items that had only differed in their μ values, so the algorithm processes fewer items. This technique can also be seen in Moore (2000) and Klein and Manning (2001a,b). Importantly,

this means that each nonterminal only has one complete item, $[j, k, B \rightarrow \star\bullet]$, for each span. This effect alone is enough to improve the runtime of Earley's to $\mathcal{O}\big(N^3|\mathcal{G}| + N^2|\mathcal{G}||\mathcal{R}|\big)$. Our §5.2 will give a version of the trick that only gets this effect, by folding the COMPLETE rule. The full version of Leermakers (1992)'s trick is subsumed by our generalized approach in §7.

While the GHR algorithm—a modified version of Earley's algorithm—is commonly known to be $\mathcal{O}\big(N^3|\mathcal{G}||\mathcal{R}|\big)$, Graham et al. (1980, §3) provide a detailed exploration of the low-level implementation of their algorithm that enables it to be run in $\mathcal{O}\big(N^3|\mathcal{G}|\big)$ time. This explanation spans 20 pages and includes techniques similar to those mentioned in §5, as well as discussion of data structures. To the best of our knowledge, these details have not been carried forward in subsequent presentations of GHR (Stolcke, 1995; Goodman, 1999). In the deduction system view, we are able to achieve the same runtime quite easily and transparently by folding both COMPLETE (§5.2) and PREDICT (§5.1).In both cases, this eliminates the pairwise interactions between all $|\mathcal{G}|$ dotted productions and all $|\mathcal{R}|$ complete productions, thereby reducing $|\mathcal{G}||\mathcal{R}|$ to $|\mathcal{G}|$.

D Correspondence Between `Earley` and `EarleyFast`

The proofs of `EarleyFast` are in one-to-one correspondence with the proofs of `Earley`.

We show the key steps in transforming between the two styles of proof. Table 3 shows the correspondence between an application of PRED and an application of PRED1 and PRED2 , while Table 4 shows the correspondence between an application of COMP and an application of COMP1 and COMP2.

E Eliminating Unary Cycles

As mentioned in §6, our weighted deduction system requires that we eliminate unary cycles from the grammar. Stolcke (1995, §4.5) addresses the problem of unary production cycles by modifying the deduction rules.[21] He assumes use of the probability semiring, where $\mathbb{W} = [0, 1]$, $\oplus = +$, and $\otimes = \times$. In that case, inverting a single $|\mathcal{N}| \times |\mathcal{N}|$ matrix suffices to compute the total weight of all rewrite sequences $A \overset{*}{\Rightarrow} B$, known as **unary chains**, for each ordered pair $A, B \in \mathcal{N}^2$.[22] His modified rules then ignore the original unary productions and refer to these weights instead.

We take a very similar approach, but instead describe it as a transformation of the weighted grammar, leaving the deduction system unchanged. We generalize from the probability semiring to any closed semiring—that is, any semiring that provides an operator $*$ to compute geometric series sums in closed form (see App. A). In addition, we improve the construction: we do not collapse all unary chains as Stolcke (1995) does, but only those subchains that can appear on cycles. This prevents the grammar size from blowing up more than necessary (recall that the parser's runtime is proportional to grammar size). For example, if the unary productions are $A_i \rightarrow A_{i+1}$ for all $1 \leq i < K$, then there is no cycle and our transformation leaves these $K - 1$ productions unchanged, rather than replacing them with $K(K-1)/2$ new unary productions that correspond to the possible chains $A_i \overset{*}{\Rightarrow} A_j$ for $1 \leq i \leq j \leq K$.

Given a weighted CFG $\mathcal{G} = \langle \mathcal{N}, \Sigma, \mathcal{R}, S, w \rangle$, consider the weighted graph whose vertices are $\mathcal{N}$ and whose weighted edges $A \rightarrow B$ are given by the unary productions $A \rightarrow B$. (This graph may include self-loops such as $A \rightarrow A$.) Its strongly connected components (SCCs) will represent unary production cycles and can be found in linear time (and thus in $\mathcal{O}(|\mathcal{G}|)$ time). For any A and B in the same SCC, $w(A \overset{*}{\Rightarrow} B) \in \mathbb{W}$ denotes the total weight of all rewrite sequences of the form $A \overset{*}{\Rightarrow} B$ (including the 0-length sequence with weight ①, if $A = B$). For an SCC of size K, there are K^2 such weights and they can be found in total time $\mathcal{O}\big(K^3\big)$ by the Kleene–Floyd–Warshall algorithm (Lehmann, 1977; Tarjan, 1981b,a). In the real semiring, this algorithm corresponds to using Gauss-Jordan elimination to invert $I - E$, where E is the weighted adjacency matrix of the SCC (rather than of the whole graph as in Stolcke (1995)). In the general case, it computes the infinite matrix sum $I \oplus E \oplus (E \otimes E) \oplus \cdots$ in closed form, with the help of the * operator of the closed semiring.

We now construct a new grammar $\mathcal{G}' = \langle \mathcal{N}', \Sigma, \mathcal{R}', \overline{S}, w' \rangle$ that has no unary cycles, as follows. For each $A \in \mathcal{N}$, our $\mathcal{N}'$ contains two nonterminals, $\overline{A}$ and $\underline{A}$. For each ordered pair of non-

[21] Johnson (2000) provides an implementation of CKY (and the inside-outside algorithm) that allows unary productions and handles unary cycles in a similar way.

[22] In a PCFG in which all rule weights are > 0, this total weight is guaranteed finite provided that all nonterminals are generating (footnote 8).

Earley	EarleyFast
PRED: $\dfrac{B \to \nu}{[j, j, B \to \bullet\, \rho]}\ [i, j, A \to \mu \bullet B\, \nu]$	PRED1: $\dfrac{}{[j, j, B \to \bullet \star]}\ [i, j, A \to \mu \bullet B\, \nu]$ PRED2: $\dfrac{B \to \nu}{[j, j, B \to \bullet\, \rho]}\ [j, j, B \to \bullet \star]$

Table 3: Any application of PRED in `Earley` has a one-to-one correspondence with an application of PRED1 and PRED2 in `EarleyFast`. Note that it is not possible for a derivation in `EarleyFast` to have $[j, j, B \to \bullet\, \rho]$ without state $[j, j, B \to \bullet \star]$ and an application of PRED1.

Earley	EarleyFast
COMP: $\dfrac{[i, j, A \to \mu \bullet B\, \nu] \quad [j, k, B \to \rho \bullet]}{[i, k, A \to \mu\, B \bullet \nu]}$	COMP2: $\dfrac{[i, j, A \to \mu \bullet B\, \nu] \quad \text{COMP1: } \dfrac{[j, k, B \to \rho \bullet]}{[j, k, B \to \star \bullet]}}{[i, k, A \to \mu\, B \bullet \nu]}$

Table 4: Any application of COMP in `Earley` has a one-to-one correspondence with an application of COMP1 and COMP2 in `EarleyFast`. Note that it is not possible for a derivation in `EarleyFast` to have $[j, k, B \to \star \bullet]$ without state $[j, k, B \to \rho \bullet]$ and an application of COMP1.

terminals $A, B \in \mathcal{N}^2$ that fall in the same SCC, $\mathcal{R}'$ contains a production $\overline{A} \to \underline{B}$ with $w'\big(\overline{A} \to \underline{B}\big) = w\Big(A \overset{*}{\Rightarrow} B\Big)$. For every rule $A \to \rho$ in $\mathcal{R}$ that is *not* of the form $A \to B$ where A and B fall in the same SCC, $\mathcal{R}'$ also contains a production $\underline{A} \to \overline{\rho}$ with $w'(\underline{A} \to \overline{\rho}) = w(A \to \rho)$, where $\overline{\rho}$ is a version of ρ in which each nonterminal B has been replaced by $\overline{B}$. Finally, as a constant-factor optimization, $\overline{A}$ and $\underline{A}$ may be merged back together if A formed a trivial SCC with no self-loop: that is, remove the weight-① production $\overline{A} \to \underline{A}$ from $\mathcal{R}'$ and replace all copies of $\overline{A}$ and $\underline{A}$ with A throughout $\mathcal{G}'$.

Of course, as Aycock and Horspool (2002) noted, this grammar transformation does change the derivations (parse trees) of a sentence, which is also true for the grammar transformation in App. F below. A derivation under the new grammar (with weight w) may represent infinitely many derivations under the old grammar (with total weight w). In principle, if the old weights were in the derivation semiring (see App. A), then w will be a representation of this infinite set. This implies that the $*$ operator in this section, and the polynomial system solver in App. F below, must be able to return weights in the derivation semiring that represent infinite context-free languages.

F Eliminating Nullary Productions

In addition to unary cycles (App. E) we must eliminate nullary productions in order to avoid cyclic proofs, as mentioned in §6. This must be done *before* eliminating unary cycles, since eliminating nullary productions can create new unary productions. Hopcroft et al. (2007, §7.1.3) explain how to do this in the unweighted case. Stolcke (1995, §4.7.4) sketches a generalization to the probability semiring, but it also uses the non-semiring operations of division and subtraction (and is not clearly correct). We therefore give an explicit general construction.

While we provide a method that handles nullary productions by modifying the grammar, it is also possible to instead modify the algorithm to allow advancing the dot over **nullable** nonterminals, i.e., nonterminals A such that the grammar allows $A \overset{*}{\Rightarrow} \varepsilon$ (Aycock and Horspool, 2002).

Our first step, like Stolcke's, is to compute the "null weight"

$$e_A \overset{\text{def}}{=} w\Big(A \overset{*}{\Rightarrow} \varepsilon\Big) \overset{\text{def}}{=} \bigoplus_{\substack{T \in \mathcal{T}^A: \\ \text{yield}(T) = \varepsilon}} w(T) \tag{7}$$

for each $A \in \mathcal{N}$. Although a closed semiring does not provide an operator for this summation, these values are a solution to the system of $|\mathcal{N}|$ polynomial equations[23]

$$e_A = \bigoplus_{(A \to B_1 \cdots B_n) \in \mathcal{R}} w(A \to B_1 \cdots B_n) \otimes \bigotimes_{i=1}^{n} e_{B_i} \tag{8}$$

In the same way, the free weights from equation (4) in §6.1 are a solution to the system

$$Z_A = \bigoplus_{(A \to \rho) \in \mathcal{R}} w(A \to \rho) \otimes \bigotimes_{i:\rho_i \in \mathcal{N}} Z_{\rho_i} \tag{9}$$

which differs only in that ρ is allowed to contain terminal symbols. In both cases, the distributive

[23] If $(A \to \varepsilon) \in \mathcal{R}$, it will be covered by the case $n = 0$.

property of semirings is being used to recursively characterize a sum over what may be infinitely many trees. A solution to system (8) must exist for the sums in equation (2) to be well-defined in the first place. (Similarly, a solution to system (9) must exist for the sums in equations (3) and (4) to be well-defined.) If there are multiple solutions, the desired sum is given by the "minimal" solution, in which as many variables as possible take on value ⓪. Often in practice the minimal solution can be found using fixed-point iteration, which initializes all free weights to ⓪ and then iteratively recomputes them via system (8) (respectively system (9)) until they no longer change (e.g., at numerical convergence). For example, this is guaranteed to work in the tropical semiring $(\mathbb{W}, \oplus, \otimes, ⓪, ①) = (\mathbb{R}_{\geq 0}, \min, +, \infty, 0)$ and more generally in ω-continuous semirings under conditions given by Kuich (1997). Esparza et al. (2007) and Etessami and Yannakakis (2009) examine a faster approach based on Newton's method. Nederhof and Satta (2008) review methods for the case of the real weight semiring $(\mathbb{W}, \oplus, \otimes, ⓪, ①) = (\mathbb{R}_{\geq 0}, +, \times, 0, 1)$.

Given the null weights $e_A \in \mathbb{W}$, we now modify the grammar as follows. We adopt the convention that for a production $A \to \rho$ that is not yet in $\mathcal{R}$, we consider its weight to be $w(A \to \rho) = ⓪$, and increasing this weight by any non-⓪ amount adds it to $\mathcal{R}$. For each nonterminal B such that $e_B \neq ⓪$, let us assume the existence of an auxiliary nonterminal $B_{\neq\varepsilon} \notin \mathcal{N}$ such that $B_{\neq\varepsilon} \overset{*}{\not\Rightarrow} \varepsilon$ but $\forall \mathbf{x} \neq \varepsilon$, $w\left(B_{\neq\varepsilon} \overset{*}{\Rightarrow} \mathbf{x}\right) = w\left(B \overset{*}{\Rightarrow} \mathbf{x}\right)$. We iterate this step: as long as we can find a production $A \to \mu\, B\, \nu$ in $\mathcal{R}$ such that $e_B \neq ⓪$, we modify it to the more restricted version $A \to \mu\, B_{\neq\varepsilon}\, \nu$ (keeping its weight), but to preserve the possibility that $B \overset{*}{\Rightarrow} \varepsilon$, we also increase the weight of the shortened production $A \to \mu\, \nu$ by $e_B \otimes w(A \to \mu\, B\, \nu)$.

A production $A \to \rho$ where ρ includes k nonterminals B with $e_B \neq ⓪$ will be gradually split up by the above procedure into 2^k productions, in which each B has been either specialized to $B_{\neq\varepsilon}$ or removed. The shortest of these productions is $A \to \varepsilon$, whose weight is $w(A \to \varepsilon) = e_A$ by equation (8). So far we have preserved all weights $w\left(A \overset{*}{\Rightarrow} \mathbf{x}\right)$, provided that the auxiliary nonterminals behave as assumed. For each A we must now remove $A \to \varepsilon$ from $\mathcal{R}$, and since A can no longer rewrite as ε, we rename all other rules $A \to \rho$ to $A_{\neq\varepsilon} \to \rho$. This closes the loop by defining the auxiliary nonterminals as desired.

Finally, since S is the start symbol, we add back $S \to \varepsilon$ (with weight e_S) as well as adding the new rule $S \to S_{\neq\varepsilon}$ (with weight ①). Thus (as in Chomsky Normal Form), the only nullary rule is now $S \to \varepsilon$, which may be needed to generate the 0-length sentence. We now have a new grammar with nonterminals $\mathcal{N}' = \{S\} \cup \{B_{\neq\varepsilon} : B \in \mathcal{N}\}$. To simplify the names, we can rename the start symbol S to S' and then drop the $_{\neq\varepsilon}$ subscripts. Also, any nonterminals that *only* rewrote as ε in the original grammar are no longer generating and can be safely removed (see footnote 8).

G Working With Left Corners

G.1 Recursive Chains in Prefix Outside Weights

As mentioned in §6.1, there is a subtle issue that arises if the grammar has left-recursive productions. Consider the left-recursive rule $B \to B\, \rho$. Using equation (5), the prefix outside weight of the predicted item $[j, j, B \to \bullet\, B\, \rho]$ will only include the weight corresponding to one rule application of $B \to B\, \rho$, but correctness demands that we account for the possibility of recursively applying $B \to B\, \rho$ as well. A well-known technique to remove left-recursion is the left-corner transform (Rosenkrantz and Lewis, 1970; Johnson and Roark, 2000). As that may lead to drastic increases in grammar size, however, we instead provide a modification of PRED1 that deals with this technical complication (which adapts Stolcke (1995, §4.5.1) to our improved deduction system and generalizes it to closed semirings). Fig. 3 provides some further intuition on the left-recursion issue.

We require some additional definitions: B is a left child of A iff there exists a rule $A \to B\, \rho$. The *reflexive* and transitive closure of the left-child relation is $\overset{*}{\Rightarrow}_L$, which was already defined in §2. A nonterminal A is said to be left-recursive if A is a nontrivial left corner of itself, i.e., if $A \overset{+}{\Rightarrow}_L A$ (meaning that $A \to B\, \rho$ and $B \overset{*}{\Rightarrow}_L A$ for some B). A grammar is left-recursive if at least one of its nonterminals is left-recursive.

To deal with left-recursive grammars, we collapse the weights of left-recursive paths similarly as we did with unary cycles (see App. E), and $\otimes$-multiply in at the PRED1 step.

We consider the left-corner multigraph: given a weighted CFG $\mathcal{G} = \langle \mathcal{N}, \Sigma, \mathcal{R}, S, w \rangle$, its vertices are $\mathcal{N}$ and its edges are given by the left-child re-

START:	$\dot{\alpha}([0, 0, B \to \bullet \star])$	$\oplus= w\left(S \overset{*}{\Rightarrow}_L B\right)$
PRED1LR:	$\dot{\alpha}([j, j, B \to \bullet \star])$	$\oplus= \dot{\alpha}([i, j, A \to \mu \bullet C\, \nu]) \otimes w\left(C \overset{*}{\Rightarrow}_L B\right) \otimes \dot{\beta}([i, j, A \to \mu \bullet C\, \nu])$
PRED2:	$\dot{\alpha}([j, j, B \to \bullet \rho])$	$\oplus= \dot{\alpha}[j, j, B \to \bullet \star]$
SCAN:	$\dot{\alpha}([i, k, A \to \mu\, a \bullet \nu])$	$\oplus= \dot{\alpha}[i, j, A \to \mu \bullet\, a\, \nu]$
COMP1:	$\dot{\alpha}([j, k, B \to \star \bullet])$	$\oplus= \dot{\alpha}[j, k, B \to \rho \bullet]$
COMP2:	$\dot{\alpha}([i, k, A \to \mu\, B \bullet \nu])$	$\oplus= \dot{\alpha}[j, j, B \to \bullet \rho]$
POS:	$\dot{\alpha}([j, j])$	$\oplus= \dot{\alpha}([i, j, A \to \mu\, a \bullet \nu]) \otimes \dot{\beta}([i, j, A \to \mu\, a \bullet \nu]) \otimes Z_\nu$

Table 5: Explicit formulas for incrementing the prefix outside weights during one-step proofs for `EarleyFast` for the general case in which the grammar may be left-recursive, as explained in App. G.1. Note that the prefix outside weights for COMP1 go unused for subsequent proof steps, and thus do not contribute to the prefix weights associated with the input string $\mathbf{x}$. The prefix outside weight for $\dot{\alpha}([j, j])$ is the desired prefix weight $w\left(S \overset{*}{\Rightarrow}_L \mathbf{x}_{0:j}\right)$.

lations, with one edge for every production. Each edge is associated with a weight equal to the weight of the corresponding production $\otimes$-times the free weights of the nonterminals on the right hand side of the production that are not the left-child. For instance, for a production $A \to B\ C\ D$, the weight of the corresponding edge in the graph will be $w(A \to B\ C\ D) \otimes Z_C \otimes Z_D$. This graph's SCCs represent the left-corner relations. For any A and B in the same SCC $w\left(A \overset{*}{\Rightarrow}_L B\right) \in \mathbb{W}$ denotes the total weight of all left-corner rewrite sequences of the form $A \overset{*}{\Rightarrow}_L B$, including the free weights needed to compute the prefix outside weights. These can, again, be found in $\mathcal{O}\left(K^3\right)$ time with the Kleene–Floyd–Warshall algorithm (Lehmann, 1977; Tarjan, 1981b,a), where K is the size of the SCC. These weights can be precomputed and have no effect on the runtime of the parsing algorithm. We replace PRED1 with the following:

$$\text{PRED1LR:}\ \frac{[i, j, A \to \mu \bullet C\, \nu]}{[j, j, B \to \bullet \star]}\ \mu \neq \varepsilon \quad C \overset{*}{\Rightarrow}_L B$$

A one-step proof of PRED1LR contributes

$$\dot{\alpha}([i, j, A \to \mu \bullet C\, \nu]) \otimes w\left(C \overset{*}{\Rightarrow}_L B\right) \otimes \dot{\beta}([i, j, A \to \mu \bullet C\, \nu]) \tag{10}$$

to the prefix outside weight $\dot{\alpha}([j, j, B \to \bullet \rho])$.

Note that the case $B = C$ recovers the standard PRED1, and such rules will always be instantiated since $\overset{*}{\Rightarrow}_L$ is reflexive. The PRED1LR rule has three side conditions (whose visual layout here is not significant). Its consequent will feed into PRED2; the condition $\mu \neq \varepsilon$ ensures that the output of PRED2 cannot serve again as a side condition to PRED1, since the recursion from C was already fully computed by the $C \overset{*}{\Rightarrow}_L B$ item. However, since this condition prevents PRED1LR from predicting anything at the start of the sentence, we must also replace the start axiom $[0, 0, S \to \bullet \star]$ with a rule that resembles PRED1 and derives the start axiom together with all its left corners:

$$\text{START:}\ \frac{}{[0, 0, B \to \bullet \star]}\ S \overset{*}{\Rightarrow}_L B$$

The final formulas for aggregating the prefix outside weights are spelled out explicitly in Table 5. Note that we did not spell out a corresponding prefix weight algorithm for `EarleyFSA`.

G.2 One-Word Lookahead

Orthogonally to App. G.1, we can optionally extend the left child relation to terminal symbols, saying that a is a left child of A if there exists a rule $A \to a\, \rho$.

The resulting extended left-corner relation (in its unweighted version) can be used to construct a side condition on PRED1 (or PRED1LR), so that at position j, it does not predict all symbols that are compatible with the left context, but only those that are *also* compatible with the next input terminal.

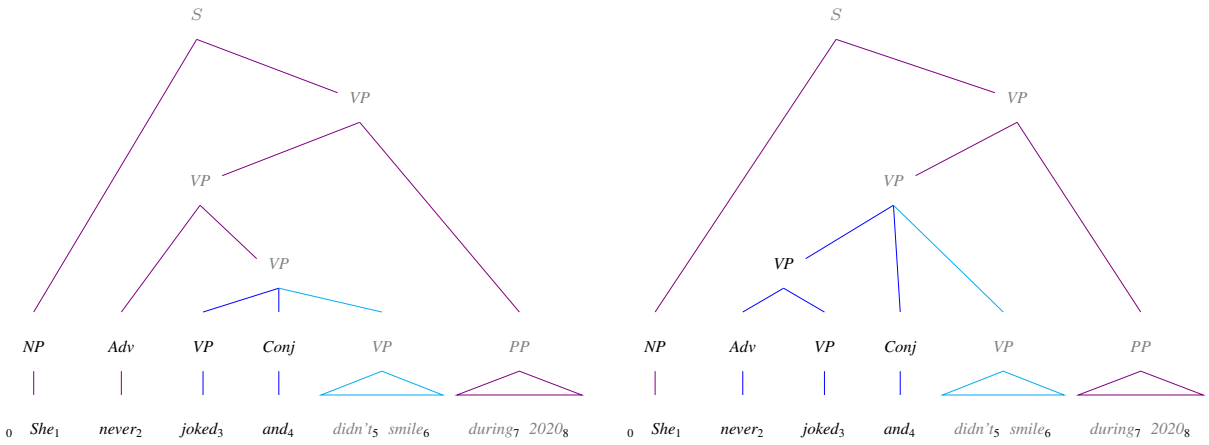

Figure 3: To find the prefix weight of the prefix "She never joked and," the algorithm must consider *all* derivations of *all* complete sentences with that prefix (§6.1). This figure shows two derivations of one such completion—the ambiguous sentence "She never joked and didn't smile during 2020." The left derivation corresponds to a proof tree for which the unique incomplete item ending at position 4 is $[2, 4, \mathit{VP} \rightarrow \mathit{VP}\ \mathit{Conj} \bullet \mathit{VP}]$ and the right derivation corresponds to a proof tree for which the unique incomplete item ending at position 4 is $[1, 4, \mathit{VP} \rightarrow \mathit{VP}\ \mathit{Conj} \bullet \mathit{VP}]$. We colorize the productions associated with these items' incomplete inside weight, future inside weight and outside weight. The prefix outside weight of $\dot{\alpha}([4, 4, \mathit{VP} \rightarrow \bullet \star])$ sums the product of these three weights over all derivations at PRED1 (equation (5)). It makes use of use of free weights to sum over all expansions of any nonterminals that were predicted to follow position 4: in these examples, Z_{VP} and Z_{PP} are included in, respectively, the future inside weight and the prefix outside weight of the incomplete antecedent item ($[2, 4, \mathit{VP} \rightarrow \mathit{VP}\ \mathit{Conj} \bullet \mathit{VP}]$ or $[1, 4, \mathit{VP} \rightarrow \mathit{VP}\ \mathit{Conj} \bullet \mathit{VP}]$). The example derivations shown both contribute one copy of $\mathit{VP} \rightarrow \mathit{VP}\ \mathit{PP}$ to the prefix outside weight. However, since that production is left-recursive, the prefix is also consistent with other completions that use it k times to produce k arbitrary future prepositional phrases, for any $k = 0, 1, 2, \ldots$. To sum over all such possibilities we provide a modified PRED1 in App. G.1. This summation over copies of $\mathit{VP} \rightarrow \mathit{VP}\ \mathit{PP}$ (and over the possible expansions of its future *PP*) is needed both when *VP* is predicted at position 1 and again when *VP* is predicted at position 2.

To be precise, PRED1 (or PRED1LR) should only predict B at position j if $[j, k, a]$ and $B \overset{*}{\Rightarrow}_L a$ (for some a). This is in fact Earley (1970)'s k-word lookahead scheme in the special case $k = 1$.

G.3 Left-Corner Parsing

Nederhof (1993) and Nederhof (1994b) describe a **left-corner parsing** technique that we could apply to further speed up Earley's algorithm. This subsumes the one-word lookahead technique of the previous section. Eisner and Blatz (2007) sketched how the technique could be derived automatically.

Normally, if B is a deeply nested left corner of C, then the item $A \rightarrow \mu \bullet C\,\nu$ will trigger a long chain of PREDICT actions that culminates in $[j, j, B \rightarrow \bullet \star]$. Unfortunately, it may not be possible for this B (or anything predicted from it) to SCAN its first terminal symbol, in which case the work has been wasted.

But recall from App. G.1 that the PRED1LR rule effectively summarizes this long chain of predictions using a precomputed weighted item $C \overset{*}{\Rightarrow}_L B$. The left-corner parsing technique simply skips the PREDICT steps and uses $C \overset{*}{\Rightarrow}_L B$ as a side condition to lazily *check after the fact* that the relevant prediction of a $\bullet$-initial rule could have been made.

PRED1 is removed, so the method never creates dotted productions of the form $A \rightarrow \mu \bullet \nu$ where $\mu = \varepsilon$—except for the start item and the items derived from it using PRED2.

In COMP2, a side condition $\mu \neq \varepsilon$ is added. For the special case $\mu = \varepsilon$, a new version of COMP2 is used in which

- $i = j$ is required,
- the first antecedent $[i, i, A \rightarrow \bullet\, B\,\nu]$ is replaced by $A \rightarrow B\,\nu$ (which ensures that $[i, i, A \rightarrow \bullet\, B\,\nu]$ is an item of `EarleyFast`),
- the side conditions $[h, i, D \rightarrow \mu' \bullet C\,\nu']$ and $C \overset{*}{\Rightarrow}_L A$ (which ensures that `EarleyFast` would have PREDICTed that item). Note that $\mu' = \varepsilon$ is possible in the case where D is the start symbol S.

The SCAN rule is split in exactly the same way into $\mu \neq \varepsilon$ and $\mu = \varepsilon$ variants.

H Execution of Weighted `EarleyFast`

Eisner (2023) presents generic strategies for executing unweighted and weighted deduction systems. We apply these here to solve the weighted recognition and prefix weight problems, by computing the weights of all items that are provable from given grammar and sentence axioms.

H.1 Execution via Multi-Pass Algorithms

The `Earley` and `EarleyFast` deduction systems are nearly acyclic, thanks to our elimination of unary rule cycles and nullary rules from the grammar. However, cycles in the left-child relation can still create deduction cycles, with $[k, k, A \to \bullet\, B\, X]$ and $[k, k, B \to \bullet\, A\, Y]$ proving each other via PRED or via PRED1 and PRED2.

Weighted deduction can be accomplished for these systems using the generic method of Eisner (2023, §7). This will detect the left-child cycles at runtime (Tarjan, 1972) and solve the weights to convergence within each strongly connected component (SCC). While solving the SCCs can be expensive in general, it is trivial in our setting since the weights of the items within an SCC do not actually depend on one another: these items serve only as side conditions for one another. Thus, any iterative method will converge immediately.

Alternatively, the deduction system becomes fully acyclic when we eliminate prediction chains as shown in App. G.1. In particular, this modified version of `EarleyFast` replaces PRED1 with PRED1LR.[24] Using this acyclic deduction system allows a simpler execution strategy: under any acyclic deduction system, a reference-counting strategy (Kahn, 1962) can be applied to find the proved items and then compute their weights in topologically sorted order (Eisner, 2023, §6).

In both cyclic and acyclic cases, the above weighted recognition strategies consume only a constant factor more time and space than their unweighted versions, across all deduction systems and all inputs.[25] For `EarleyFast` and its acyclic version, this means the runtimes are $\mathcal{O}(N|\mathcal{G}|)$ for a class of "bounded-state" grammars, $\mathcal{O}\big(N^2|\mathcal{G}|\big)$ for unambiguous grammars, and $\mathcal{O}\big(N^3|\mathcal{G}|\big)$ for general grammars (as previewed in the abstract and §1). The space requirements are respectively $\mathcal{O}(N|\mathcal{G}|)$, $\mathcal{O}\big(N^2|\mathcal{G}|\big)$, and $\mathcal{O}\big(N^2|\mathcal{G}|\big)$. The same techniques apply to `EarleyFSA`, replacing $|\mathcal{G}|$ with $|\mathcal{M}|$.

H.2 One-Pass Execution via Prioritization

For the acyclic version of the deduction system (App. G.1), an alternative strategy is to use a prioritized agenda to visit the items of the acyclic deduction system in some topologically sorted order (Eisner, 2023, §5). This may be faster in practice than the generic reference-counting strategy because it requires only one pass instead of two. It also remains space-efficient. On the other hand, it requires a priority queue, which adds a term to the asymptotic runtime (worsening it in some cases such as bounded-state grammars).

We must associate a **priority** $\pi(V)$ with each item V such that if U is an antecedent or side condition in some rule that proves V, then $\pi(U) < \pi(V)$. Below, we will present a nontrivial prioritization scheme in which the priorities implicitly take the form of lexicographically ordered tuples.

These priorities can easily be converted to integers in a way that preserves their ordering. Thus, a bucket queue (Dial, 1969) or an integer priority queue (Thorup, 2000) can be used (see Eisner (2023, §5) for details). The added runtime overhead[26] is $\mathcal{O}(M)$ for the bucket queue or $\mathcal{O}(M' \log\log M')$ for the integer priority queue, where $M = \mathcal{O}\big(N^2|\mathcal{G}|\big)$ is the number of distinct priority levels in the set of *possible* items, and $M' \leq M$ is the number of distinct priority levels of the *actually* proved items, which depends on the grammar and input sentence.

For `EarleyFast` with the modifications of App. G.1, we assign the minimum priority to all of the axioms. All other items have one of six forms:

1. $[j, k, B \to \rho\, \bullet]$ (antecedent to COMP1, POS)
2. $[j, k, B \to \star\, \bullet]$ (rightmost antecedent to COMP2)
3. $[j, k, B \to \mu\, \bullet\, \nu]$ where $\mu \neq \varepsilon, \nu \neq \varepsilon$ (leftmost antecedent to PRED1LR, SCAN, POS)
4. $[k, k]$ (antecedent to nothing)
5. $[k, k, B \to \bullet\, \star]$ (antecedent to PRED2)
6. $[k, k, B \to \bullet\, \rho]$ (leftmost antecedent to SCAN, COMP2)

[24] Recall that eliminating the left-child cycles in advance in this way is needed when one wants to compute weights of the form $w(V) = (\dot{\beta}(V), \dot{\alpha}(V))$, in which case the items in an SCC do *not* merely serve as side conditions for one another. The weighted deduction formalism of Eisner (2023) is flexible enough to handle cyclic rules that would correctly define these pairwise weights in terms of one another, but solving the SCCs would no longer be fast.

[25] Excluding the time to solve the SCCs in the cyclic case; but for us, the statement holds even when including that time.

[26] Under the Word RAM model of computation and assuming that priorities fit into a single word.

The relative priorities of these items are as follows:

- Items with smaller k are visited sooner (left-to-right processing).
- Among items with the same k, items with $j < k$ are visited before items with $j = k$. Thus, the leftmost antecedent of PRED1LR precedes its consequent.
- Among items with the same k and with $j < k$, items with larger j are visited sooner. Thus, the rightmost antecedent of COMP2 precedes its consequent in the case $i < j$, where a narrower item is used to build a wider one.
- Among items of the first two forms with the same k and the same $j < k$, B is visited sooner than A if $A \overset{*}{\Rightarrow} B$. This ensures that the rightmost antecedent of COMP2 precedes its consequent in the case $i = j$ and $\nu = \varepsilon$, which completes a unary constituent.

  To facilitate this comparison, one may assign integers to the nonterminals according to their height in the unweighted graph whose vertices are $\mathcal{N}$ and whose edges $A \to B$ correspond to the unary productions $A \to B$. (This graph is acyclic once unary cycles have been eliminated by the method of App. E.)
- Remaining ties are broken in the order of the numbered list above. This ensures that the antecedents of COMP1, POS, and PRED2 precede their consequents, and the rightmost antecedent of COMP2 precedes its consequent in the case $i = j$ and $\nu \neq \varepsilon$, which starts a non-unary constituent.

To understand the flow of information, notice that the 6 specific items in the numbered list above would be visited in the order shown.

H.3 Pseudocode for Prioritized Algorithms

For concreteness, we now give explicit pseudocode that runs the rules to build all of the items in the correct order. This may be easier to implement than the above reductions to generic methods. It is also slightly more efficient than App. H.2, due to exploiting some properties of our particular system.

Furthermore, in this section we handle `EarleyFast` as well as its acyclic modification. When the flag p is set to *true*, we carry out the acyclic version, which replaces PRED1 with PRED1LR and START (App. G.1), and also includes POS (§6.1) to find prefix weights.

The algorithm pops (dequeues) and processes items in the same order as App. H.2 (when p is *true*), except that in this version, axioms of the form $B \to \rho$ and $[k-1, k, a]$ are never pushed (enqueued) or popped but are only looked up in indices. Similarly, the $[j, j]$ items (used to find prefix weights) are never pushed or popped but only proved. Thus, none of these items need priorities.

When an item U is popped, our pseudocode invokes only deduction rules for which U might match the *rightmost* antecedent (which could be a side condition), or in the case of SCAN or PRED1LR, the *leftmost* antecedent. In all cases, the other antecedents are either axioms or have lower priorities. While we do not give pseudocode for each rule, invoking a rule on U always checks first whether U actually does match the relevant antecedent. If so, it looks up the possible matches for its other antecedents from among the axioms and the previously proved items. This may allow the rule to prove consequents, which it adds to the queues and indices as appropriate (see below).

The main routine is given as Alg. 1. A queue iteration such as "**for** $k \in \mathcal{Q}$: ..." iterates over a collection that may change during iteration; it is shorthand for "**while** $\mathcal{Q} \neq \emptyset$: { $k = \mathcal{Q}$.pop(); ... }."

We maintain a dictionary (the **chart**) that maps items to their weights. Each time an item V is proved by some rule, its weight $w(V)$ is updated accordingly, as explained in §6 and Table 5. The weight is $\dot{\beta}(V)$ or $(\dot{\beta}(V), \dot{\alpha}(V))$ according to whether p is *false* or *true*.

Alg. 1 writes $\mathcal{C}(\textit{pattern})$ to denote the set of all provable items (including axioms) that match *pattern*. This set will have previously been computed and stored in an **index** dedicated to the specific invocation of $\mathcal{C}(\textit{pattern})$ in the pseudocode. The index is another dictionary, with the previously bound variables of the pattern serving as the key. The pseudocode for individual rules also uses indices, to look up antecedents.

When an item V is *first* proved by a rule and added to the chart, it is also added to all of the appropriate sets in the indices. Prioritization ensures that we do not look up a set until it has converged.

Each dictionary may be implemented as a hash table, in which case lookup takes expected $O(1)$ time under the Uniform Hashing Assumption. An array may also be used for guaranteed $O(1)$ access,

although its sparsity may increase the algorithm's asymptotic space requirements.[27]

What changes when p is *false*, other than a few of the rules? Just one change is needed to the prioritization scheme of App. H.2. The `EarleyFast` deduction system is cyclic, as mentioned in App. H.1, so in this case, we cannot enforce $\pi(U) < \pi(V)$ when U and V are an antecedent and consequent of the same rule. We will only be able to guarantee $\pi(U) \leq \pi(V)$, where the $=$ case arises only for PRED1 and PRED2. To achieve this weaker prioritization, we modify our tiebreaking principle from App. H.2 (when p is *false*) to say that for a given k, all items of the last two forms have *equal* priority and thus may be popped in any order.[28] When a rule proves a consequent that has the same priority as one of its antecedents, it is possible that the consequent had popped previously. In our case, this happens only for the rule PRED1, so crucially, it does not matter if the new proof changes the consequent's weight—this consequent is used only as a side condition (to PRED2) so its weight is ignored. However, to avoid duplicate work, we must take care to avoid re-pushing the consequent now that it has been proved again.[29]

Rather than place all the items on a single queue that is prioritized lexicographically as in App. H.2, we use a collection of priority queues that are combined in the pseudocode to have the same effect. They are configured and maintained as follows.

- $\mathcal{Q}$ is a priority queue of distinct positions $k \in \{0, \ldots, N\}$, which pop in increasing order. k is added to it upon proving an item of the form $[\cdot, k, \cdot]$. Initially $\mathcal{Q} = \{0\}$ due to the start axiom $[0, 0, S \to \bullet\star]$.
- For each $k \in \mathcal{Q}$, $\mathcal{P}_k$ is a priority queue of distinct positions $j \in \{0, \ldots, k\}$, which pop in decreasing order except that k itself pops last. j is added to it upon proving an item of the form $[j, k, \cdot]$. Initially $\mathcal{P}_0 = \{0\}$ due to the start axiom.

Algorithm 1 `EarleyFast` with priority queues

```
1:  function EARLEYFAST(G, x, p)
2:    add G, x axioms to dictionaries and queues
3:    if p : START()                          ▷ apply START (App. G.1)
4:    for k ∈ Q :                             ▷ that is: while Q ≠ ∅, pop into k
5:      for j ∈ P_k :
6:        for B ∈ N_jk :
7:          for U ∈ C([j, k, B → ρ •]) :      ▷ form 1
8:            COMP1(U); POS(U)
9:          for U ∈ C([j, k, B → ⋆ •]) :      ▷ form 2
10:           COMP2(U)
11:       for U ∈ C([j, k, B → μ • ν], μ≠ε≠ν) :
12:         SCAN(U); POS(U)                   ▷ form 3
13:         if p : PRED1LR(U) else PRED1(U)
14:     if p and k > 0 :                      ▷ form 4
15:       now w([k, k]) = prefix weight of x_{0:k}
16:     if p :
17:       for U ∈ C([k, k, B → • ⋆]) :        ▷ form 5
18:         PRED2(U)
19:       for U ∈ C([k, k, B → • ρ]) :        ▷ form 6
20:         SCAN(U)
21:     else
22:       for U ∈ S_k :                       ▷ forms 5 and 6
23:         ▷ prediction may push new items onto S_k
24:         PRED1(U); PRED2(U); SCAN(U)
25:   for U ∈ C([0, |x|, S → ⋆ •]) :
26:     return w(U)                           ▷ weight of goal item
27:   return ⓪                                ▷ goal item has not been proved
```

- For each $j \in \mathcal{P}_k$ with $j < k$, $\mathcal{N}_{jk}$ is a priority queue of distinct nonterminals $B \in \mathcal{N}$, which pop in the height order described in App. H.2 above. B is added to it upon proving an item of the form $[j, k, B \to \rho \bullet]$.
- If p is *false*, then for each $k \in \mathcal{Q}$, $\mathcal{S}_k$ is a queue of all proved items of the form $[k, k, B \to \bullet \star]$ or $[k, k, B \to \bullet \rho]$. These items have equal priority so may pop in any order (e.g., LIFO). Initially $\mathcal{S}_0$ contains just the start axiom.

Transitive consequents added later to a queue always have $\geq$ priority than their antecedents that have already popped, so the minimum priority of the queue increases monotonically over time. This monotone property is what makes bucket queues viable in our setting (see Eisner, 2023, §5). In general, our priority queues are best implemented as bucket queues if they are dense and binary heaps or integer priority queues if they are sparse.

[27] Its sparsity need not increase the runtime requirements, however: an *uninitialized* array can be used to simulate an initialized array with constant overhead. Higham and Schenk (1993) attribute this technique to computer science folklore.

[28] Formerly, all items of form 5 already had equal priority, and so did all items of form 6, but the former priority was strictly lower. This worked because there were no prediction chains.

[29] Specifically, this discussion implies that in general, when a consequent may have the same priority as an antecedent, we must check whether it has *ever* been pushed onto the queue, not whether it is currently on the queue. Luckily, this is easily done by checking whether it is a key in the chart.

I Binarized `EarleyFSA`

Table 6 gives a version of `EarleyFSA` in which the ternary deduction rules SCAN, COMP1 and COMP2 have been binarized using the fold transform, as promised in §7.

- The SCAN1 and SCAN2 rules, which replace SCAN, introduce and consume new intermediate items of the form $[i, j, q \overset{a}{\rightsquigarrow} \star]$. The SCAN1 rule sums over possible start positions j for word a. This is only advantageous in the case of lattice parsing (see footnote 7), since for string parsing, the only possible choice of j is $k - 1$.
- In a similar vein, COMP2A and COMP2B introduce and consume new intermediate items $[i, j, \star \overset{A}{\rightsquigarrow} q]$. The COMP2A rule aggregates different items from i to j that are looking for a B constituent to their immediate right, summing over their possible current states q.
- Similarly, COMP1A introduces new intermediate items that sum over possible final states q'.
- We did not bother to binarize the ternary rule FILTER, as there is no binarization that provides an asymptotic speed-up.

There are different ways to binarize inference rules, and in Table 6 we have chosen to binarize SCAN and COMP2 in complementary ways. Our binarization of SCAN is optimized for the common case of a dense WFSA and a sparse sentence, where state q allows many terminal symbols a but the input allows only one (as in string parsing) or a few. SCAN1 finds just the symbols a allowed by the input and SCAN2 looks up only those out-arcs from q. Conversely, our binarization of COMP2 is optimized for the case of a sparse WFSA and a dense parse table: COMP2A finds the small number of incomplete constituents over $[i, j]$ that are looking for a B, and COMP2B looks those up when it finds a complete B constituent, just like `EarleyFast`.

It is possible to change each of these binarizations. In particular, binarizing SCAN by first combining $[i, j, q]$ with $q \overset{a}{\rightsquigarrow} q'$ (analogously to COMP2A) would be useful when parsing a large or infinite lattice—such as the trie implicit in a neural language model—with a constrained grammar (Shin et al., 2021; Fang et al., 2023).

J Handling Nullary and Unary Productions in an FSA

As for `EarleyFast`, `EarleyFSA` (§7) requires elimination of nullary productions. We can handle nullary productions by directly adapting the construction of App. F to the WFSA case. Indeed, the WFSA version is simpler to express. For each arc $q \overset{B}{\rightsquigarrow} q'$ such that $B \in \mathcal{N}$ and $e_B \neq \textcircled{0}$, we replace the B label of that arc with $B_{\neq\varepsilon}$ (preserving the arc's weight), and add a new arc $q \overset{\varepsilon}{\rightsquigarrow} q'$ of weight e_B. We then define a new WFSA $\mathcal{M}' = (\mathcal{M} \cap \neg\mathcal{M}_{\text{bad}}) \cup \mathcal{M}_{\text{good}}$, where $\mathcal{M}_{\text{bad}}$ is an unweighted FSA that accepts exactly those strings of the form $\widehat{A}$ (i.e., nullary productions), $\neg$ takes the unweighted complement, and $\mathcal{M}_{\text{good}}$ is a WFSA that accepts exactly strings of the form $\widehat{S}$ (with weight e_S) and $S_{\neq\varepsilon}\widehat{S}$ (with weight $\textcircled{1}$). As this construction introduces new ε arcs, it should precede the elimination of ε-cycles.

Notice that in the example of App. F where a production $A \to \rho$ was replaced with up to $2^k - 1$ variants, the WFSA construction efficiently shares structure among these variants. It adds at most k edges at the first step and at most doubles the total number of states through intersection with $\neg\mathcal{M}_{\text{bad}}$.

Similarly, we can handle unary productions by directly adapting the construction of App. E to the WFSA case. We first extract all weighted unary rules by intersecting $\mathcal{M}$ with the unweighted language $\{B\widehat{A} : A, B \in \mathcal{N}\}$ (and determinizing the result so as to combine duplicate rules). Exactly as in App. E, we construct the unary rule graph and compute its SCCs along with weights $w\left(A \overset{*}{\Rightarrow} B\right)$ for all A, B in the same SCC. We modify the WFSA by underlining all hatted nonterminals $\widehat{A}$ and overlining all nonterminals B. Finally, we define our new WFSA grammar $(\mathcal{M} \cap \neg\mathcal{M}_{\text{bad}}) \cup \mathcal{M}_{\text{good}}$. Here $\mathcal{M}_{\text{bad}}$ is an unweighted FSA that accepts exactly those strings of the form $\overline{B}\underline{\widehat{A}}$ and $\mathcal{M}_{\text{good}}$ is a WFSA that accepts exactly strings of the form $\underline{B}\widehat{\overline{A}}$ such that A, B are in the same SCC, with weight $w\left(A \overset{*}{\Rightarrow} B\right)$.

Following each construction, nonterminal names can again be simplified as in Apps. E and F.

Finally, §7 mentioned that we must eliminate ε-cycles from the FSA. The algorithm for doing so (Mohri, 2002) is fundamentally the same as our method for eliminating unary rule cycles from a CFG (App. E), but now it operates on the graph

Domains $i, j, k \in \{0, \ldots, N\} \quad A \in \mathcal{N} \quad a \in \Sigma \quad q, q' \in \mathcal{Q}$

Items $[i, j, q] \quad [i, j, q?] \quad [i, j, a] \quad [i, j, A \to \bullet \star] \quad [i, j, A \to \star \bullet] \quad [i, j, q \overset{a}{\rightsquigarrow} \star] \quad [i, j, \star \overset{A}{\rightsquigarrow} q]$

$$\underbrace{q \in \mathcal{I} \quad q' \in \mathcal{F} \quad q \overset{a}{\rightsquigarrow} q' \quad q \overset{A}{\rightsquigarrow} q' \quad q \overset{A}{\rightsquigarrow} \star \quad q \overset{*\widehat{A}}{\rightsquigarrow} \star \quad q \overset{\widehat{B}}{\rightsquigarrow} \mathcal{F}}_{\text{WFSA items}}$$

Axioms $[k-1, k, x_k], \forall k \in \{1, \ldots, N\} \qquad [0, 0, S \to \bullet \star]$ WFSA items derived from the WFSA grammar (see §7)

Goals $[0, N, S \to \star \bullet]$

Rules

PRED1: $\dfrac{}{[j, j, B \to \bullet \star]}\ [i, j, q] \quad q \overset{B}{\rightsquigarrow} \star$

PRED2: $\dfrac{q \in \mathcal{I}}{[j, j, q?]}$

SCAN1: $\dfrac{[i, j, q] \quad [j, k, a]}{[i, k, q \overset{a}{\rightsquigarrow} \star]}$

SCAN2: $\dfrac{[i, k, q \overset{a}{\rightsquigarrow} \star] \quad q \overset{a}{\rightsquigarrow} q'}{[i, k, q'?]}$

EPSILON: $\dfrac{[i, j, q] \quad q \overset{\varepsilon}{\rightsquigarrow} q'}{[i, j, q'?]}$

COMP1A: $\dfrac{q \overset{\widehat{B}}{\rightsquigarrow} q' \quad q' \in \mathcal{F}}{q \overset{\widehat{B}}{\rightsquigarrow} \mathcal{F}}$

COMP1B: $\dfrac{[j, k, q] \quad q \overset{\widehat{B}}{\rightsquigarrow} \mathcal{F}}{[j, k, B \to \star \bullet]}$

COMP2A: $\dfrac{[i, j, q] \quad q \overset{B}{\rightsquigarrow} q'}{[i, j, \star \overset{B}{\rightsquigarrow} q']}$

COMP2B: $\dfrac{[i, j, \star \overset{B}{\rightsquigarrow} q'] \quad [j, k, B \to \star \bullet]}{[i, k, q'?]}$

FILTER: $\dfrac{[j, k, q?]}{[j, k, q]}\ [j, j, A \to \bullet \star] \quad q \overset{*\widehat{A}}{\rightsquigarrow} \star$

Table 6: Improvement of `EarleyFSA` (Table 2) in which SCAN, COMP1 and COMP2 have been binarized using a fold transform. Since COMP1A does not depend on the input it can in practice be run during preprocessing, just like the rules that derive other WFSA items such as $q \overset{*\widehat{A}}{\rightsquigarrow} \star$. See the main text (App. I) for a discussion of alternative binarization schemes.

whose edges are ε-arcs of the FSA, rather than the graph whose edges are unary rules of the CFG.

K Non-Commutative Semirings

We finally consider the case of non-commutative weight semirings, where the order of multiplication becomes significant.

In this case, in the product (1) that defines the weight of a derivation tree T, the productions should be multiplied in the order of a pre-order traversal of T.

In §3, when we recursively defined the weight $w(d_V)$ of a proof, we took a product over the above-the-bar antecedents of a proof rule. These should be multiplied in the same left-to-right order that is shown in the rule. Our deduction rules are carefully written so that under these conventions, the resulting proof weight matches the weight (1) of the corresponding CFG derivation.

For the same reason, the same left-to-right order should be used in §3 when computing the inside probability $\dot{\beta}(V)$ of an item.

Eliminating nullary productions from a weighted CFG (App. F) is not in general possible in non-commutative semirings. However, if the grammar has no nullary productions or is converted to an FSA before eliminating nullary productions (App. J), then weighted parsing may remain possible.

What goes wrong? The construction in App. F unfortunately reorders the weights in the product (1). Specifically, in the production $A \to \mu\, B\, \nu$, the product should include the weight e_B *after* the weights in the μ subtrees, but our construction made it part of the weight of the modified production $A \to \mu\, \nu$ and thus moved it *before* the μ subtrees. This is incorrect when $\mu \neq \varepsilon$ and $\otimes$ is non-commutative.

The way to rescue the method is to switch to using WFSA grammars (§7). The WFSA grammar breaks each rule up into multiple arcs, whose weights variously fall before, between, and after the weights of its children. When defining the weight of a derivation under the WFSA grammar, we do not simply use a pre-order traveral as in equation (1). The definition is easiest to convey informally through an example. Suppose a derivation tree for $A \overset{*}{\Rightarrow} \mathbf{x}$ uses a WFSA path at the root that

accepts $BC\widehat{A}$ with weight w. Recursively let w_B and w_C be the weights of the child subderivations, rooted at B and C. Then the overall weight of the derivation of A will not be $w \otimes w_B \otimes w_C$ (prefix order), but rather $w_1 \otimes w_B \otimes w_2 \otimes w_C \otimes w_3$. Here we have factored the path weight w into $w_1 \otimes w_2 \otimes w_3$, which are respectively the weights of the subpath up through B (including the initial-state weight), the subpath from there up through C, and the subpath from there to the end (including the final-state weight).

When converting a CFG to an equivalent WFSA grammar (footnote 14), the rule weight always goes at the *start* of the rule so that the weights are unchanged. However, the nullary elimination procedure for the WFSA (App. J) is able to replace unweighted nonterminals in the *middle* of a production with weighted ε-arcs. This is the source of its extra power, as well as its greater simplicity compared to App. F.

It really is not possible to fully eliminate nulls within the simpler weighted CFG formalism. Consider an unambiguous weighted CFG whose productions are $S \to a\,S\,A, S \to b\,S\,B, S \to c, A \to \varepsilon, B \to \varepsilon$, with respective weights w_a, w_b, w_c, w_A, w_B. Then a string $\mathbf{x} = abbc$ will have $Z_{\mathbf{x}}$ given by the mirrored product $w_a \otimes w_b \otimes w_b \otimes w_c \otimes w_B \otimes w_B \otimes w_A$. Within our weighted CFG formalism, there is no way to include the final weights $w_B \otimes w_B \otimes w_A$ if we are not allowed to have null constituents in those positions.

Even with WFSAs, there is still a problem—in the non-commutative case, we cannot eliminate unary rule cycles (App. J). If we had built a binary A constituent with weight w, then a unary CFG rule $A \to A$ with weight w_1 required us to compute the total weight of all derivations of A, by taking a summation of the form $w \oplus (w_1 \otimes w) \oplus (w_1 \otimes w_1 \otimes w) \oplus \cdots$. This factors as $(① \oplus w_1 \oplus (w_1 \otimes w_1) \oplus \cdots) \otimes w$, and unary rule cycle elimination served to precompute the parenthesized sum, which was denoted as $w\left(A \overset{*}{\Rightarrow} A\right)$, and record it as the weight of a new rule $\overline{A} \to \underline{A}$. However, in the non-commutative case, the WFSA path corresponding to $A \to A$ might start with w_1 and end with w_2. In that case, the necessary summation has the form $w \oplus (w_1 \otimes w \otimes w_2) \oplus (w_1 \otimes w_1 \otimes w \otimes w_2 \otimes w_2) \oplus \cdots$. Unfortunately this cannot be factored as before, so we cannot precompute the infinite sums as before.[30] The construction in App. J assumed that we could extract weighted unary rules from the WFSA, with a single consolidated weight at the start of each rule—but consolidating the weight in that way required commutativity.

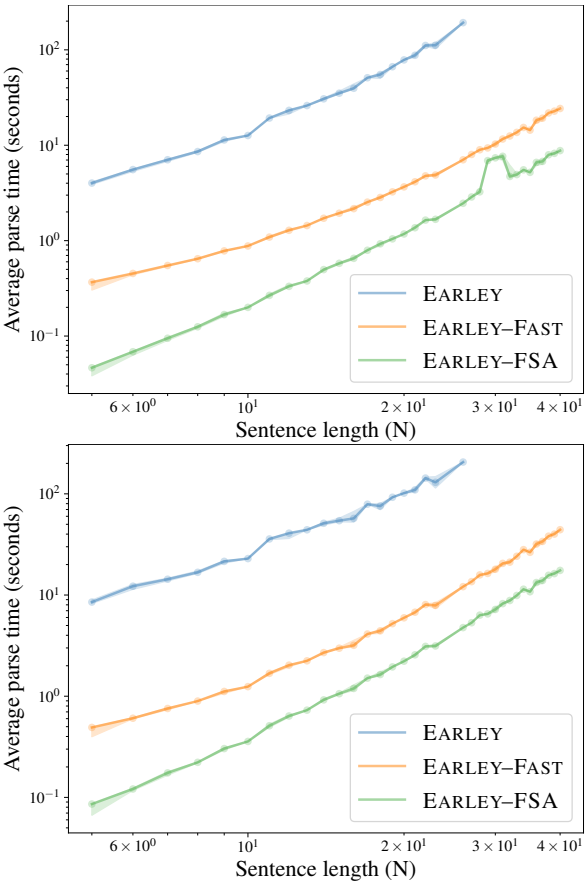

Figure 4: Average parse time per sentence for 100 randomly selected sentences of 5–40 words on the M2 grammar (left) and PM2 grammar (right). As all these algorithms are worst-case cubic in N, each curve on these log-log plots is bounded above by a line of slope 3, but the lower lines have better grammar constants. The experiment was conducted using a Cython implementation on an Intel(R) Core(TM) i7-7500U processor with 16GB RAM.

L Runtime Experiment Results

More details on the experiments of §8 appear in Fig. 4.

[30] A related problem would appear in trying to generalize the left-corner rewrite weights in App. G.1 to the non-commutative case.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Sec 9

☐ A2. Did you discuss any potential risks of your work?
Not applicable. We do not foresee any direct risks of our work.

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Sec 0 and sec 1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☒ Did you use or create scientific artifacts?

Left blank.

☐ B1. Did you cite the creators of artifacts you used?
No response.

☐ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
No response.

☐ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
No response.

☐ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
No response.

☐ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
No response.

☐ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
No response.

C ☑ Did you run computational experiments?

Sec 7 and app I

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Sec 7

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Sec 7

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
App I

☐ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Not applicable. Left blank.

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.