Efficient Algorithms for Recognizing Weighted Tree-Adjoining Languages

Alexandra Butoi[1] **Tim Vieira**[1] **Ryan Cotterell**[1] **David Chiang**[2]
[1]ETH Zürich [2]University of Notre Dame
{alexandra.butoi, ryan.cotterell}@inf.ethz.ch
dchiang@nd.edu tim.f.vieira@gmail.com

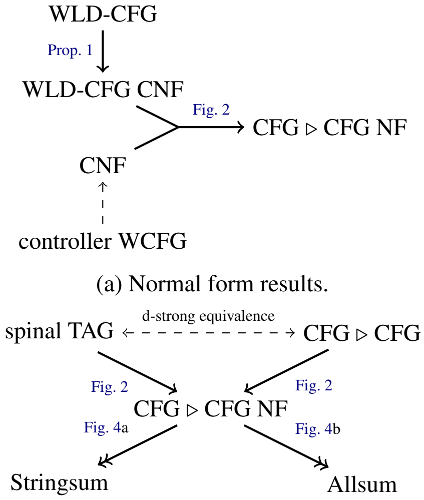
Abstract

The class of tree-adjoining languages can be characterized by various two-level formalisms, consisting of a context-free grammar (CFG) or pushdown automaton (PDA) controlling another CFG or PDA. These four formalisms are equivalent to tree-adjoining grammars (TAG), linear indexed grammars (LIG), pushdown-adjoining automata (PAA), and embedded pushdown automata (EPDA). We define semiring-weighted versions of the above two-level formalisms, and we design new algorithms for computing their stringsums (the weight of all derivations of a string) and allsums (the weight of all derivations). From these, we also immediately obtain stringsum and allsum algorithms for TAG, LIG, PAA, and EPDA. For LIG, our algorithm is more time-efficient by a factor of $\mathcal{O}(n|\mathcal{N}|)$ (where n is the string length and $|\mathcal{N}|$ is the size of the nonterminal set) and more space-efficient by a factor of $\mathcal{O}(|\Gamma|)$ (where Γ is the size of the stack alphabet) than the algorithm of Vijay-Shanker and Weir (1989). For EPDA, our algorithm is both more space-efficient and time-efficient than the algorithm of Alonso et al. (2001) by factors of $\mathcal{O}(|\Gamma|^2)$ and $\mathcal{O}(|\Gamma|^3)$, respectively. Finally, we give the first PAA stringsum and allsum algorithms.

1 Introduction

Weir (1992) introduced a hierarchy of formal languages whose first level ($\mathcal{L}_1$) is the class of context-free languages and second level ($\mathcal{L}_2$) is the class of tree-adjoining languages. Just as context-free languages can be characterized by both context-free grammars and pushdown automata, tree-adjoining languages are characterized by multiple formalisms as well, including tree-adjoining grammars (TAG; Joshi et al., 1975), linear indexed grammars (LIG; Gazdar, 1988), embedded pushdown automata (EPDA; Vijay-Shanker, 1987), and pushdown-adjoining automata (PAA; Butoi et al., 2023).

Tree-adjoining languages can further be characterized through the mechanism of *control* (Weir, 1992), which yields various *two-level* formalisms. Specifically, we have shown that CFGs controlled by CFGs, CFGs controlled by PDAs, PDAs controlled by CFGs, and PDAs controlled by PDAs are equivalent to TAG, LIG, PAA, and EPDA, respectively, in a strict sense called *d-strong* equivalence (Butoi et al., 2023).



(a) Normal form results.

(b) Stringsum and allsum results.

Figure 1: Overview of the results presented in this paper. Solid lines are new results shown in the paper; dashed lines are old results. An arrow $X \rightarrow Y$ means "X can be converted to Y"; an arrow $X \twoheadrightarrow Y$ means "X has an algorithm for computing Y". We only show the conversions and algorithms for CFG ▷ CFG; the results for the other three two-level formalisms are analogous.

When designing statistical parsers for tree-adjoining formalisms, it is often useful to consider their semiring-weighted generalizations. In this paper, we introduce semiring-weighted versions of the above two-level formalisms, and we give new, more efficient algorithms for computing stringsums (the total weight of all derivations of a string) in these formalisms.

Lang (1974) gives a recognition algorithm for (what we call) simple PDAs, which can pop and push at most one stack symbol. We have shown that this algorithm is suboptimal, and that (what we call) top-down PDAs, which always pop exactly one stack symbol, allow computing stringsums more

Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, pages 5394–5416
December 6-10, 2023 ©2023 Association for Computational Linguistics

efficiently (Butoi et al., 2022).

Existing definitions of LIG (Vijay-Shanker and Weir, 1989; Joshi et al., 1991; Kanazawa, 2014) are equivalent to CFGs controlled by simple PDAs, and existing recognition algorithms (e.g., Vijay-Shanker and Weir, 1989) have the same limitation that makes Lang's algorithm suboptimal. By using a top-down PDA as a controller instead, we obtain a new stringsum algorithm that is more space-efficient and runs asymptotically faster. Additionally, we obtain an algorithm for allsums (the total weight of all derivations) that is more space-efficient.

All stringsum algorithms that operate directly on EPDAs that we are aware of (Alonso et al., 2000, 2001) are designed for specific types of EPDAs, failing to take advantage of the structure of computation in top-down PDAs. We design a new stringsum algorithm for a subclass of EPDA that is equivalent to a top-down PDA controlled by a top-down PDA, and we obtain both time and space improvements over these previous algorithms. Additionally, we design a more space-efficient EPDA allsum algorithm.

Our algorithms assume that CFGs are given in Chomsky normal form, and for PDAs, we define a new normal form that is exactly analogous to the Chomsky normal form. We show that applying these normal forms to controllers and controllees induces, for free, normal forms for the two-level formalisms and for TAG, LIG, PAA, and EPDA. The conversions into the normal forms for all these formalisms become simpler than direct conversions, as they only require CFG or PDA conversions. We leave extensions of our stringsum and allsum algorithms to general CFGs/PDAs for future work.

The main contributions of this paper are:

- Semiring-weighted versions of the formalisms CFG ▷ CFG (§3), PDA ▷ CFG, CFG ▷ PDA, and PDA ▷ PDA (App. B.1).
- Normal forms for these formalisms that arise, for free, from the normal forms of the controller/-controllee CFGs/PDAs (§3.2 and App. B.2).
- Stringsum algorithms for CFG ▷ CFG (§4), PDA ▷ CFG, and PDA ▷ PDA (App. C.2) that are more time-efficient or space-efficient than the existing algorithms for their equivalent LIG and EPDA.
- The first stringsum algorithm for CFG ▷ PDA, and therefore, for PAA (App. C.2).
- Algorithms for computing allsums in these two-level formalisms (§5 and App. D).

2 Preliminaries

Let $[i{:}j]$ denote the sequence of integers $(i, \ldots, j)$. If $\boldsymbol{s}$ is a string, we write $|\boldsymbol{s}|$ for the length of $\boldsymbol{s}$, $\boldsymbol{s}_i$ for the i^{th} symbol of $\boldsymbol{s}$, and $\boldsymbol{s}_{(i:j]}$ for the substring $\boldsymbol{s}_{i+1} \cdots \boldsymbol{s}_j$.

2.1 Semiring-Weighted Languages

Throughout this paper, we assume that $\mathcal{W} = (W, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a commutative semiring. We also sometimes assume that $\mathcal{W}$ is ω-continuous, which makes it possible to take (countably) infinite sums. Please see App. A.1 for definitions of these terms. Readers not familiar with semirings may safely assume that $W = \mathbb{R}_{\geq 0} \cup \{\infty\}$, $\mathbf{0} = 0$, $\mathbf{1} = 1$, and $\oplus$ and $\otimes$ are the usual addition and multiplication operations, respectively.

Definition 1. *An* ***alphabet*** Σ *is a non-empty finite set of* ***symbols****. A* ***language*** *over* Σ *is a subset of* Σ*'s* ***Kleene closure*** $\Sigma^* \overset{\text{def}}{=} \bigcup_{n \geq 0} \Sigma^n$*. A* ***weighted language*** *over* Σ *with weights from* $\mathcal{W} = (W, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ *is a mapping from* Σ^* *to* W*.*

This paper considers *weighted* formalisms, where the weights are taken from $\mathcal{W}$. Rather than just producing or recognizing strings from a formal language, these formalisms define a weighting function over Σ^*. When $\mathcal{W}$ is the boolean semiring $(\{0, 1\}, \vee, \wedge, 0, 1)$, we recover the usual notion of a formal language defined as a set of strings.

2.2 Weighted Context-Free Grammars

Definition 2. *A* ***weighted context-free grammar*** *(WCFG) over a semiring* $\mathcal{W} = (W, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ *is a tuple* $\mathbf{G} = (\mathcal{N}, \Sigma, \mathcal{R}, \mathrm{w}, S)$*, where* $\mathcal{N}$*,* Σ *are finite sets of nonterminal symbols and terminal symbols, respectively,* $\mathcal{R} \subseteq \mathcal{N} \times (\Sigma \cup \mathcal{N})^*$ *is a finite set of productions,* $\mathrm{w} \colon \mathcal{R} \to W$ *is a production-weighting function, and* $S \in \mathcal{N}$ *is the start symbol.*

We write a production $(X, \boldsymbol{\alpha}) \in \mathcal{R}$ *as* $X \to \boldsymbol{\alpha}$*, and if* $\mathrm{w}(X \to \boldsymbol{\alpha}) = w$*, we write* $X \xrightarrow{w} \boldsymbol{\alpha}$*.*

A WCFG produces strings by starting from the symbol S and repeatedly replacing the leftmost nonterminal X with the right-hand side of a production $X \to \boldsymbol{\alpha}$ until no more nonterminals are left. In the following definitions, let $\mathbf{G} = (\mathcal{N}, \Sigma, \mathcal{R}, \mathrm{w}, S)$ be a WCFG.

Definition 3. *If* $\boldsymbol{\alpha} = \boldsymbol{\beta}_1 X \boldsymbol{\beta}_2$ *and* $\boldsymbol{\alpha}' = \boldsymbol{\beta}_1 \boldsymbol{\gamma} \boldsymbol{\beta}_2$ *are sequences of terminals and nonterminals of* $\mathbf{G}$*, where* $\boldsymbol{\beta}_1 \in \Sigma^*$ *and* $\boldsymbol{\beta}_2 \in (\Sigma \cup \mathcal{N})^*$*, and* $p =$

*$(X \xrightarrow{w} \gamma)$ is a production of $\mathbf{G}$, we write $\boldsymbol{\alpha} \overset{p}{\Rightarrow} \boldsymbol{\alpha}'$ to denote that $\boldsymbol{\alpha}$ **derives** $\boldsymbol{\alpha}'$ in one step using production p.*

Definition 4. *A **partial derivation** in $\mathbf{G}$ from $\boldsymbol{\alpha}_0$ to $\boldsymbol{\alpha}_n$ is a sequence of steps $d = \boldsymbol{\alpha}_0 \overset{p_1}{\Longrightarrow} \cdots \overset{p_n}{\Longrightarrow} \boldsymbol{\alpha}_n$. We write $\boldsymbol{\alpha}_0 \overset{*}{\Rightarrow} \boldsymbol{\alpha}_n$ to assert that some partial derivation from $\boldsymbol{\alpha}_0$ to $\boldsymbol{\alpha}_n$ exists, or to denote the set of all such partial derivations. If $\boldsymbol{\alpha}_0 = S$ and $\boldsymbol{\alpha}_n = \boldsymbol{s} \in \Sigma^*$, we call d a **derivation** of $\boldsymbol{s}$, or that $\mathbf{G}$ **derives** $\boldsymbol{s}$.*

*The **weight** of d is the product of its production weights,*

$$\mathbf{w}(d) \overset{\text{def}}{=} \bigotimes_{i=1}^{n} \mathrm{w}(p_i).$$

We denote by $\mathcal{D}(\mathbf{G}, \boldsymbol{s})$ the set of all derivations in $\mathbf{G}$ of $\boldsymbol{s}$ and by $\mathcal{D}(\mathbf{G})$ the set of all derivations in $\mathbf{G}$.

Definition 5. *The **stringsum** $\mathbf{w}(\mathbf{G}, \boldsymbol{s})$ of a string $\boldsymbol{s}$ under $\mathbf{G}$ is the total weight of all derivations in $\mathbf{G}$ for $\boldsymbol{s}$,*

$$\mathbf{w}(\mathbf{G}, \boldsymbol{s}) \overset{\text{def}}{=} \bigoplus_{d \in \mathcal{D}(\mathbf{G}, \boldsymbol{s})} \mathbf{w}(d).$$

Definition 6. *The **allsum** $\mathbf{w}(\mathbf{G})$ of $\mathbf{G}$ is the total weight of all its derivations,*

$$\mathbf{w}(\mathbf{G}) \overset{\text{def}}{=} \bigoplus_{d \in \mathcal{D}(\mathbf{G})} \mathbf{w}(d).$$

3 Semiring-Weighted CFG ▷ CFG

Weir (1992) defined a hierarchy of formal languages and showed that its second level, which is commonly known as the class of tree-adjoining languages, can be obtained through the mechanism of control, using a CFG (the controllee) whose derivations are controlled by another CFG (the controller). In previous work (Butoi et al., 2023), we extended this mechanism to PDAs, both as controllers and controllees, and obtained four distinct formalisms by mixing a controller CFG or PDA with a controllee CFG or PDA. In this paper, we use *semiring-weighted* versions of these formalisms. We give a formal definition of a weighted CFG ▷ CFG here (see App. B.1 for the other three definitions).

We denote such a grammar by $\mathbf{G}_1 \triangleright \mathbf{G}_2$, where $\mathbf{G}_1$ is the controller and $\mathbf{G}_2$ is the controllee. Additionally, we use symbols $X, Y, Z, \ldots$ and $a, b, c, \ldots$ for nonterminals and terminals of the controllee, and $\mathsf{A}, \mathsf{B}, \mathsf{C}, \ldots$ and $\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots$ for nonterminals and terminals of the controller.

3.1 Definition

A weighted labeled distinguished CFG (WLD-CFG) is simply a WCFG where each production has a label, and its right-hand side has one “distinguished” occurrence of a nonterminal symbol. These two extensions allow its derivations to be controlled by another formalism.

Definition 7. *A **weighted labeled distinguished context-free grammar** (WLD-CFG) over a semiring $\mathcal{W} = (W, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a tuple $\mathbf{G} = (\mathcal{N}, \Sigma, L, \mathcal{R}, \mathrm{w}, S)$, where*

- *$\mathcal{N}$, Σ, and L are finite sets of nonterminal symbols, terminal symbols, and labels, respectively,*
- *$\mathcal{R} \subseteq L \times \mathbb{N} \times \mathcal{N} \times (\mathcal{N} \cup \Sigma)^*$ is a finite set of productions,*
- *$\mathrm{w} \colon \mathcal{R} \rightarrow W$ is a production-weighting function, and*
- *$S \in \mathcal{N}$ is the start symbol.*

If $(\ell, i, A, \beta_1 \cdots \beta_n)$ is a production in $\mathcal{R}$, we must have either $i = 0$, which we write as

$$\ell : A \rightarrow \beta_1 \cdots \beta_n$$

or $1 \le i \le n$ and $\beta_i \in \mathcal{N}$, which we write as

$$\ell : A \rightarrow \beta_1 \cdots \beta_{i-1} \breve{\beta_i} \beta_{i+1} \cdots \beta_n.$$

Weir (1992) gave two definitions of a derivation in an LD-CFG (the controllee) controlled by another CFG (the controller). In his first definition, the controllee nonterminals keep a record of the productions used, called a **control word** (a string in L^*). At the end of the derivation, each control word is checked for membership in the language of the controller. In his second definition, the controllee nonterminals run a controller derivation. When the controller generates a label $\ell \in L$, it causes the controllee to apply production ℓ. We use the latter definition, which allows one to think of the controller and the controllee as a single grammar, merging their productions into a single set of rules.

For any sets $\mathcal{X}$ and $\mathcal{Y}$, we define $\mathcal{X}[\mathcal{Y}] = \{X[Y] \mid X \in \mathcal{X}, Y \in \mathcal{Y}\}$, where $X[Y]$ is just another way of writing the ordered pair of X and Y. If $\boldsymbol{\alpha} = X_1 \cdots X_k \in \mathcal{X}^*$ is a sequence of nonterminals, we use the notation $\boldsymbol{\alpha}[Y] = X_1[Y] \cdots X_k[Y]$ for any $Y \in \mathcal{Y}$.

To make the following definition more readable, we assume that the controller's right-hand sides are either a single label or a string of nonterminals, and that the controllee right-hand sides are either a single terminal or a string of nonterminals with exactly one distinguished position. It would be possible, but more tedious, to write the definition for the general case.

Definition 8. *Let $\mathbf{G}_1$ be a WCFG with nonterminals $\mathcal{N}_1$ and terminals L, called the* ***controller****, and let $\mathbf{G}_2$ be a WLD-CFG with nonterminals $\mathcal{N}_2$ and labels L, called the* ***controllee****. Then $\mathbf{G}_1 \triangleright \mathbf{G}_2$ is a rewriting system with the following rules:*

- *If $(\mathsf{A} \xrightarrow{w} \boldsymbol{\beta})$ is a production of $\mathbf{G}_1$ where $\boldsymbol{\beta} \in \mathcal{N}_1^*$, then $\mathbf{G}_1 \triangleright \mathbf{G}_2$ has a rule $X[\mathsf{A}\cdot\cdot] \xrightarrow{w} X[\boldsymbol{\beta}\cdot\cdot]$ for each $X \in \mathcal{N}_2$.*
- *If $(\mathsf{A} \xrightarrow{w_1} \ell)$ is a production of $\mathbf{G}_1$ and $(\ell\colon X \xrightarrow{w_2} \boldsymbol{\alpha}_1 \breve{Y} \boldsymbol{\alpha}_2)$ is a production of $\mathbf{G}_2$, then $\mathbf{G}_1 \triangleright \mathbf{G}_2$ has a rule $X[\mathsf{A}\cdot\cdot] \xrightarrow{w_1 \otimes w_2} \boldsymbol{\alpha}_1[\mathsf{S}]\, Y[\cdot\cdot]\, \boldsymbol{\alpha}_2[\mathsf{S}]$.*

A derivation in $\mathbf{G}_1 \triangleright \mathbf{G}_2$ starts with $S[\mathsf{S}]$. If there is a rule $p = (X[\mathsf{A}\cdot\cdot] \xrightarrow{w} \boldsymbol{\alpha}_1 Y[\boldsymbol{\beta}_1\cdot\cdot]\boldsymbol{\alpha}_2)$, then for any $\boldsymbol{\alpha}_0 \in \Sigma^*$, $\boldsymbol{\beta}_2 \in \mathcal{N}_2^*$, and $\boldsymbol{\alpha}_3 \in (\Sigma \cup \mathcal{N}_1[\mathcal{N}_2^*])^*$, we write

$$\boldsymbol{\alpha}_0 X[\mathsf{A}\boldsymbol{\beta}_2]\boldsymbol{\alpha}_3 \overset{p}{\Rightarrow} \boldsymbol{\alpha}_0\boldsymbol{\alpha}_1\, Y[\boldsymbol{\beta}_1\boldsymbol{\beta}_2]\boldsymbol{\alpha}_2\boldsymbol{\alpha}_3.$$

Similarly, if there is a rule $p = (X[\mathsf{A}] \xrightarrow{w} \boldsymbol{\alpha})$, then for any $\boldsymbol{\alpha}_0 \in \Sigma^*$, and $\boldsymbol{\alpha}_3 \in (\Sigma \cup \mathcal{N}_1[\mathcal{N}_2^*])^*$, we write

$$\boldsymbol{\alpha}_0 X[\mathsf{A}]\boldsymbol{\alpha}_3 \overset{p}{\Rightarrow} \boldsymbol{\alpha}_0\boldsymbol{\alpha}\boldsymbol{\alpha}_3.$$

We write $\overset{*}{\Rightarrow}$ to denote a derivation with zero or more steps, as in Def. 4. If $S[\mathsf{S}] \overset{*}{\Rightarrow} \boldsymbol{s} \in \Sigma^*$, we say that $\mathbf{G}_1 \triangleright \mathbf{G}_2$ ***derives*** *$\boldsymbol{s}$.*

Example 1. *Consider the following $\mathbf{G}_1 \triangleright \mathbf{G}_2$ that generates the language $\{\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n\mathrm{d}^n \mid n \in \mathbb{N}\}$. The controller is a WCFG $\mathbf{G}_1$ with the start symbol S_1 and the set of productions*

$$\mathcal{R}_1 = \left\{ \begin{array}{ll} \mathsf{S}_1 \xrightarrow{\mathbf{1}} \mathsf{T}\mathsf{L}_3 & \\ \mathsf{T} \xrightarrow{\mathbf{1}} \mathsf{L}_1\mathsf{T}\mathsf{L}_2 & \\ \mathsf{T} \xrightarrow{\mathbf{1}} \varepsilon & \\ \mathsf{L}_i \xrightarrow{\mathbf{1}} \ell_i & (i \in [1:3]) \\ \mathsf{S}_1 \xrightarrow{\mathbf{1}} \ell_i & (i \in [4:7]) \end{array} \right\}$$

and the controllee is a WLD-CFG $\mathbf{G}_2$ with start symbol S_2 and the set of productions

$$\mathcal{R}_2 = \left\{ \begin{array}{ll} \ell_1\colon \mathrm{S}_2 \xrightarrow{\mathbf{1}} \mathrm{A}\breve{\mathrm{S}}_2\mathrm{D} & \ell_4\colon \mathrm{A} \xrightarrow{\mathbf{1}} \mathrm{a} \\ \ell_2\colon \mathrm{S}_2 \xrightarrow{\mathbf{1}} \mathrm{B}\breve{\mathrm{S}}_2\mathrm{C} & \ell_5\colon \mathrm{B} \xrightarrow{\mathbf{1}} \mathrm{b} \\ \ell_3\colon \mathrm{S}_2 \xrightarrow{\mathbf{1}} \varepsilon & \ell_6\colon \mathrm{C} \xrightarrow{\mathbf{1}} \mathrm{c} \\ & \ell_7\colon \mathrm{D} \xrightarrow{\mathbf{1}} \mathrm{d} \end{array} \right\}.$$

Below is a derivation of the string aabbccdd.

$$\begin{aligned} \mathrm{S}_2[\mathsf{S}_1] &\Rightarrow \mathrm{S}_2[\mathsf{T}\mathsf{L}_3] \\ &\Rightarrow \mathrm{S}_2[\mathsf{L}_1\mathsf{T}\mathsf{L}_2\mathsf{L}_3] \\ &\Rightarrow \mathrm{A}[\mathsf{S}_1]\mathrm{S}_2[\mathsf{T}\mathsf{L}_2\mathsf{L}_3]\mathrm{D}[\mathsf{S}_1] \\ &\Rightarrow \mathrm{a}\mathrm{S}_2[\mathsf{T}\mathsf{L}_2\mathsf{L}_3]\mathrm{D}[\mathsf{S}_1] \\ &\Rightarrow \mathrm{a}\mathrm{S}_2[\mathsf{L}_1\mathsf{T}\mathsf{L}_2\mathsf{L}_2\mathsf{L}_3]\mathrm{D}[\mathsf{S}_1] \\ &\overset{*}{\Rightarrow} \mathrm{aa}\mathrm{S}_2[\mathsf{L}_2\mathsf{L}_2\mathsf{L}_3]\mathrm{D}[\mathsf{S}_1]\mathrm{D}[\mathsf{S}_1] \\ &\Rightarrow \mathrm{aa}\mathrm{B}[\mathsf{S}_1]\mathrm{S}_2[\mathsf{L}_2\mathsf{L}_3]\mathrm{C}[\mathsf{S}_1]\mathrm{D}[\mathsf{S}_1]\mathrm{D}[\mathsf{S}_1] \\ &\Rightarrow \mathrm{aab}\mathrm{S}_2[\mathsf{L}_2\mathsf{L}_3]\mathrm{C}[\mathsf{S}_1]\mathrm{D}[\mathsf{S}_1]\mathrm{D}[\mathsf{S}_1] \\ &\overset{*}{\Rightarrow} \mathrm{aabbccdd}. \end{aligned}$$

Analogous definitions for when the controller is a WPDA and/or the controllee is a WLD-PDA are given in App. B.1.

3.2 Normal Form

In this section, we define a normal form for CFG ▷ CFG that will help us design fast stringsum and allsum algorithms. Interestingly, this normal form arises naturally from the normal forms of the controllee and the controller. Analogous normal forms are derived for PDA ▷ CFG, CFG ▷ PDA, and PDA ▷ PDA in App. B.2.3.

Definition 9. *A WCFG is in* ***Chomsky normal form*** *if all of its productions are of one of the following types: (1) $S \to \varepsilon$, (2) $X \to a$, or (3) $X \to YZ$, where $S, X, Y, Z \in \mathcal{N}$ and $a \in \Sigma$. Moreover, S does not appear on the right-hand side of any production.*

A WLD-CFG is in Chomsky normal form if all of its productions are of type (1) or (2) above, (3a) $X \to \breve{Y}Z$, or (3b) $X \to Y\breve{Z}$.

Proposition 1. *For any WCFG $\mathbf{G}_1$ with weights in an ω-continuous semiring, there is a WCFG in Chomsky normal form that defines the same weighted language as $\mathbf{G}_1$.*

For any WLD-CFG $\mathbf{G}_2$ with weights in an ω-continuous semiring, there is a WLD-CFG in Chomsky normal form that is equivalent to $\mathbf{G}_2$.

$\mathbf{G}_1$	$\mathbf{G}_2$	$\mathbf{G}_1 \triangleright \mathbf{G}_2$	name
$\mathsf{S} \xrightarrow{w_1} \ell$	$\ell\colon S \xrightarrow{w_2} \varepsilon$	$S[\mathsf{S}] \xrightarrow{w_1 \otimes w_2} \varepsilon$	(epsilon)
$\mathsf{A} \xrightarrow{w_1} \ell$	$\ell\colon X \xrightarrow{w_2} a$	$X[\mathsf{A}] \xrightarrow{w_1 \otimes w_2} a$	(terminal)
$\mathsf{A} \xrightarrow{w_1} \ell$	$\ell\colon X \xrightarrow{w_2} \breve{Y} Z$	$X[\mathsf{A}\cdot\cdot] \xrightarrow{w_1 \otimes w_2} Y[\cdot\cdot]Z[\mathsf{S}]$	(left pop)
$\mathsf{A} \xrightarrow{w_1} \ell$	$\ell\colon X \xrightarrow{w_2} Y \breve{Z}$	$X[\mathsf{A}\cdot\cdot] \xrightarrow{w_1 \otimes w_2} Y[\mathsf{S}]Z[\cdot\cdot]$	(right pop)
$\mathsf{A} \xrightarrow{w_1} \mathsf{BC}$		$X[\mathsf{A}\cdot\cdot] \xrightarrow{w_1} X[\mathsf{BC}\cdot\cdot]$	(push)

Figure 2: Normal form of CFG ▷ CFG. Each row shows a rule from $\mathbf{G}_1$ (in Chomsky normal form) and possibly a rule from $\mathbf{G}_2$ (in Chomsky normal form) and the resulting normal-form rule of $\mathbf{G}_1 \triangleright \mathbf{G}_2$ along with its name.

Proof. See App. B.2.1. □

Recall that a CFG ▷ CFG is defined as a LD-CFG whose derivations are controlled by another CFG. When we consider only W(LD-)CFGs in Chomsky normal form, we obtain, for free, a normal form for CFG ▷ CFG only by mixing their rules, as shown in Fig. 2. For the epsilon rule, in principle the left-hand side could be $S[\mathsf{A}]$ for any A, but such rules would never be used.

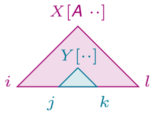
Due to d-strong equivalence, this normal form also induces a normal form for TAG. But, in order to convert a TAG into the normal form, one needs to first extract the controller and the controllee as shown by Butoi et al. (2023), then convert these to the Chomsky normal form, merge their rules, and convert them back to a TAG, also shown by Butoi et al. (2023) (see App. E for a complexity analysis of these transformations). Although these transformations add some extra complexity, the conversion into the normal form becomes simpler as it only requires (LD-)CFG conversions, rather than a direct TAG conversion. We leave a direct conversion for future work.

4 Computing Stringsums

In this section, we give a deduction system for computing stringsums of a particular string $\boldsymbol{s}$ in CFG ▷ CFG in normal form. Stringsums of a string $\boldsymbol{s}$ can be computed analogously in PDA ▷ CFG, CFG ▷ PDA, and PDA ▷ PDA; we show the deduction systems for these in App. C.2. Due to d-strong equivalence, these deduction systems can also be used for spinal TAG, LIG, EPDA, and spinal PAA by first converting them to their equivalent two-level formalism. Our algorithms have improved space requirements and run asymptotically faster than the best algorithms for LIG and EPDA. Moreover, it is the only stringsum algorithm for PAA that we are aware of.

4.1 Pop Computations

In order to compute stringsums efficiently, we decompose derivations into shareable smaller parts. Our decomposition is based on a concept that has been extensively used in the literature (Kuhlmann et al., 2011; Butoi et al., 2022) for decomposing runs of pushdown automata, namely pop computations. In the context of WPDAs, a pop computation is a partial derivation that has the overall effect of popping a single stack symbol, leaving the rest of the stack unchanged (Butoi et al., 2022). We can define pop computations of CFG ▷ CFG with a similar interpretation: partial derivations that "pop" a single leftmost nonterminal symbol, leaving the rest of the nonterminals unchanged. We define pop computations of CFG ▷ CFG formally below. The pop computations of PDA ▷ CFG, CFG ▷ PDA, and PDA ▷ PDA are defined similarly; we present these in App. C.1.

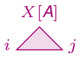
Definition 10. *Let $\mathbf{G}_1$ be a controller WCFG and $\mathbf{G}_2$ be a WLD-CFG, with nonterminal alphabets $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. Let $\boldsymbol{s} \in \Sigma^*$ be a string of length $|\boldsymbol{s}| = n$. A* ***pop computation*** *of $\mathbf{G}_1 \triangleright \mathbf{G}_2$ of type* 

, where $0 \le i \le j \le k \le l \le n$, $X, Y \in \mathcal{N}_2$, and $\mathsf{A} \in \mathcal{N}_1$, is a pair of partial derivations (d_1, d_2), where

$$d_1 \in (X[\mathsf{A}] \overset{*}{\Rightarrow} \boldsymbol{s}_{(i:j)}\, Y[\varepsilon]\boldsymbol{\Upsilon}_2)$$

$$d_2 \in (\boldsymbol{\Upsilon}_2 \overset{*}{\Rightarrow} \boldsymbol{s}_{(k:l)}).$$

The weight of the pop computation is $\mathbf{w}(d_1, d_2) = \mathbf{w}(d_1) \otimes \mathbf{w}(d_2)$.

A ***pop computation*** *of type* 

is a partial derivation $X[\mathsf{A}] \overset{*}{\Rightarrow} \boldsymbol{s}_{(i:j)}$.

In words, a pop computation of a CFG ▷ CFG pops a single controller symbol and derives a string possibly with a *gap*. Fig. 3 shows a visual representation of a pop computation of a CFG ▷ CFG.

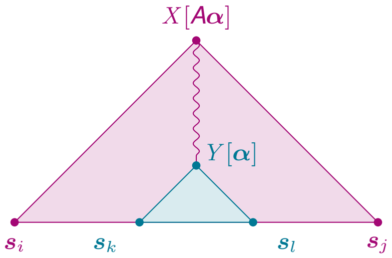


Figure 3: A pop computation of a CFG ▷ CFG is a partial derivation possibly with a gap (the teal part). First, $X[\mathsf{A}\alpha]$ derives $s_{(i:k]}$ and $Y[\alpha]$, popping one nonterminal A. Then $Y[\alpha]$ derives the gap ($s_{(k:l]}$). Finally, the purple part of the partial derivation resumes, deriving $s_{(l:j]}$. The weight of the pop computation is the weight of the purple part of the partial derivation.

4.2 Deduction System

We compute stringsums of CFG ▷ CFG efficiently by exploiting the decomposable structure of the pop computations. We give the stringsum algorithm as a weighted deduction system (Goodman, 1999), shown in Fig. 4a.

Items. The deductive system has two types of items, which correspond to the two types of pop computations (one with a gap and one without a gap). The weights of these items are the total weights of the pop computations of those types.

Deduction rules. We distinguish several types of pop computations based on their first production and include a deduction rule for each. The weight of each pop computation can be derived recursively from the weights of shorter pop computations. The weight of each deduction rule is the weight of the production in the side condition. Notice that the deduction system only shows how to derive further items, leaving the order in which these items should be derived by an algorithm unspecified.

Goal item. The goal item is $\overset{S[\mathsf{S}]}{{}_{0}\triangle_{n}}$, which stands for all derivations from $S[\mathsf{S}]$ to s, where $|s| = n$. Its weight is the total weight of all such derivations, which is exactly the stringsum.

4.3 Correctness

We prove the correctness of the deduction system for CFG ▷ CFG using the following theorem.

Theorem 1. *Let* $\mathbf{G}_1$ *be a controller WCFG with nonterminal alphabet* $\mathcal{N}_1$ *and start symbol* S, *and* $\mathbf{G}_2$ *a WLD-CFG with nonterminal alphabet* $\mathcal{N}_2$. *Let* $s \in \Sigma^*$ *be a string of length* $|s| = n$.

(a) The weight w *of an item* $\overset{X[\mathsf{A}]}{{}_{i}\triangle_{j}}$, *where* $X \in \mathcal{N}_2$, *and* $0 \leq i \leq j \leq n$, *is the total weight of all pop computations of this type.*

(b) The weight of an item $\overset{X[\mathsf{A}\,\cdot\cdot]}{{}_{i}\underset{j\quad k}{\triangle^{Y[\cdot\cdot]}}{}_{l}}$, *where* $\mathsf{A} \in \mathcal{N}_1$, $X, Y \in \mathcal{N}_2$, *and* $0 \leq i \leq j \leq k \leq l \leq n$, *is the total weight of all pop computations of this type.*

Proof. Define the *span* of item $\overset{X[\mathsf{A}]}{{}_{i}\triangle_{j}}$ to be $j - i$, and that of item $\overset{X[\mathsf{A}\,\cdot\cdot]}{{}_{i}\underset{j\quad k}{\triangle^{Y[\cdot\cdot]}}{}_{l}}$ to be $j - i + l - k$. We proceed by induction on the span ℓ.

Base Case. When $\ell = 1$, the pop computation consists of a single step $X[\mathsf{A}] \overset{p}{\Rightarrow} a$. The weight of item $\overset{X[\mathsf{A}]}{{}_{i-1}\triangle_{i}}$ is $\mathrm{w}(p)$ according to inference rule (terminal).

Inductive Step. Assume that the statement holds for pop computations with span at most $(\ell - 1)$ and consider pop computations with span ℓ. The weight of all pop computations of a certain type is the sum of various cases, distinguished by their first production.

Starting with left pop rules. Any pop computation starting with $X[\mathsf{A}]$ and a left pop production p has the form $(X[\mathsf{A}] \overset{p}{\Rightarrow} Y[\varepsilon]Z[\mathsf{S}], Z[\mathsf{S}] \overset{*}{\Rightarrow} s_{(j:k]})$.

By the inductive hypothesis, the weight w_1 of item $\overset{Z[\mathsf{S}]}{{}_{j}\triangle_{k}}$ is the total weight of all partial derivations such that $Z[\mathsf{S}] \overset{*}{\Rightarrow} s_{(j:k]}$. By distributivity, the total weight of all pop computations of the above form is $\mathbf{w}(p) \otimes w_1$. This weight is added to the weight of the item $\overset{X[\mathsf{A}\,\cdot\cdot]}{{}_{i}\underset{k\quad l}{\triangle^{Y[\cdot\cdot]}}{}_{o}}$ in inference rule (left pop).

Since the rules are symmetric, a similar argument can be used for pop computations that start with right pop rules.

Starting with push rules. Any pop computation starting with $X[\mathsf{A}]$ and a push production p has the

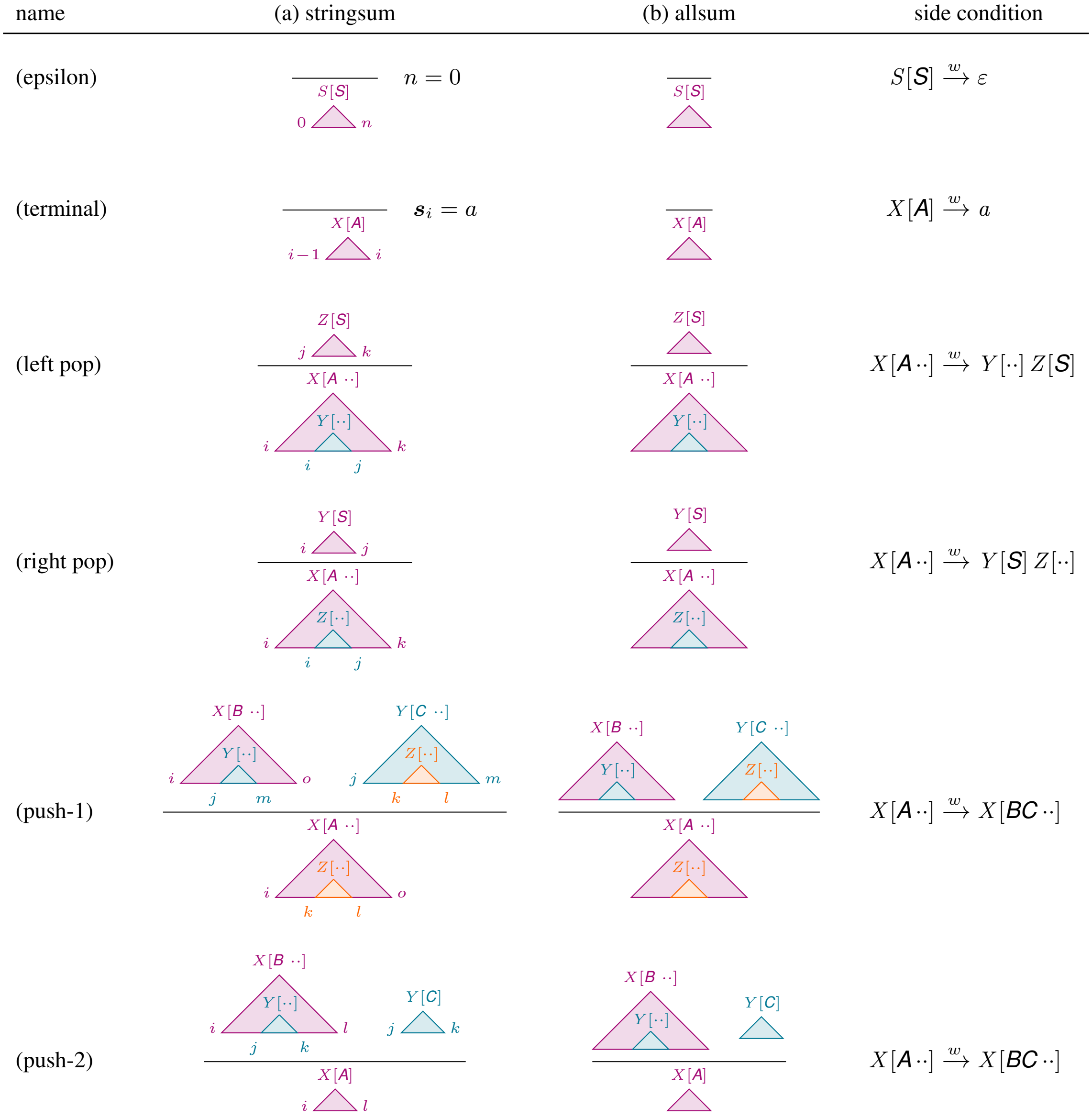
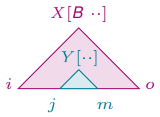
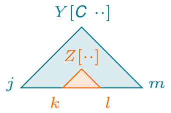
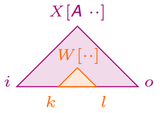


Figure 4: Deductive systems for computing stringsums and allsums of CFG ▷ CFG.

form (d_1, d_2), where

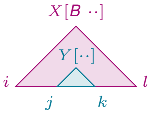
$$d_1 = X[A] \xRightarrow{p} X[BC] \xRightarrow{*} s_{(i:j)}\,Y[C]\,\Upsilon_2 \xRightarrow{*} s_{(i:j)}s_{(j:k)}\,Z[\varepsilon]\,\Upsilon_1\Upsilon_2,$$

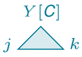
$$d_2 = \Upsilon_1\Upsilon_2 \xRightarrow{*} s_{(l:m)}\Upsilon_2 \xRightarrow{*} s_{(l:m)}s_{(m:o)}.$$

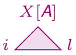
By the inductive hypothesis, the weight w_1 of item  is the weight of all pop computations $(X[B] \xRightarrow{*} s_{(i:j)}\,Y[\varepsilon]\,\Upsilon_2, \Upsilon_2 \xRightarrow{*} s_{(m:o)})$. Similarly, the weight w_2 of item  is the total weight of all pop computations $(Y[C] \xRightarrow{*} s_{(j:k)}\,Z[\varepsilon]\,\Upsilon_1, \Upsilon_1 \xRightarrow{*} s_{(l:m)})$. By distributivity, the total weight of all such partial derivations is $\mathbf{w}(p) \otimes w_1 \otimes w_2$. This weight is added to the weight of the item  in inference rule (push-1).

Any pop computation starting with $X[A]$ and a push production p has the form

$$\begin{aligned} X[A] \overset{p}{\Rightarrow} X[BC] &\overset{*}{\Rightarrow} \boldsymbol{s}_{(i:j)}\, Y[C]\boldsymbol{\Upsilon}_2 \\ &\overset{*}{\Rightarrow} \boldsymbol{s}_{(i:j)}\boldsymbol{s}_{(j:k)}\boldsymbol{\Upsilon}_2 \\ &\overset{*}{\Rightarrow} \boldsymbol{s}_{(i:j)}\boldsymbol{s}_{(j:k)}\boldsymbol{s}_{(k:l)}. \end{aligned}$$

By the inductive hypothesis, the weight w_1 of item 

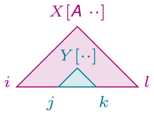
is the weight of all pop computations $(X[B] \overset{*}{\Rightarrow} \boldsymbol{s}_{(i:j)}\, Y[\varepsilon]\boldsymbol{\Upsilon}_2, \boldsymbol{\Upsilon}_2 \overset{*}{\Rightarrow} \boldsymbol{s}_{(k:l)})$, while the weight w_2 of item 

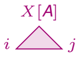
is the weight of all pop computations $Y[C] \overset{*}{\Rightarrow} \boldsymbol{s}_{(j:k)}$. By distributivity, the total weight of all such partial derivations is $\mathbf{w}(p) \otimes w_1 \otimes w_2$. This weight is added to the weight of the item 

in inference rule (push-2).

□

4.4 Complexity Analysis

Let $\mathcal{X}_1$ and $\mathcal{X}_2$ be the sets of nonterminals or stack symbols of the controllers and of the controllees, respectively. An algorithm based on one of our deductive systems would would need to store a weight for each item 

and 

, for all $i, j, k, l \in [0{:}n]$, $A \in \mathcal{X}_1$, and $X, Y \in \mathcal{X}_2$, giving a space complexity of $\mathcal{O}(n^4|\mathcal{X}_1||\mathcal{X}_2|^2)$ and $\mathcal{O}(n^2|\mathcal{X}_1||\mathcal{X}_2|)$, respectively. Overall we get a space complexity of $\mathcal{O}(n^4|\mathcal{X}_1|^2|\mathcal{X}_2|)$.

Computing the weight of each new item requires in the worst case inference rule (push-1), iterating over indices $i, j, k, l, m, n \in [0{:}n]$, and symbols $X, Y, Z \in \mathcal{X}_2$ and $A, B, C \in \mathcal{X}_1$. This gives a runtime of $\mathcal{O}(n^6|\mathcal{X}_1|^3|\mathcal{X}_2|^3)$.

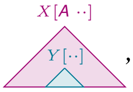
4.5 Comparison with Existing Algorithms

Vijay-Shanker and Weir (1989) designed a recognition (which is essentially computing stringsums in the boolean semiring) algorithm for a type of LIG that is d-strongly equivalent to a CFG in Chomsky normal form controlled by a simple PDA. Our stringsum algorithm is more space-efficient than theirs by a factor of $|\Gamma|$ and more time-efficient by a factor of $n|\mathcal{N}|$, where n is the string length, $|\Gamma|$ is the size of the stack alphabet, and $|\mathcal{N}|$ is the size of the alphabet of nonterminals. Their algorithm could be improved using a trick similar to the one used by Butoi et al. (2022), resulting in an algorithm with the same runtime as ours. But, in order to do so, it is necessary to store additional items, which increases the space complexity by another factor of $|\Gamma|$. Additionally, on the type of LIG used by Vijay-Shanker and Weir, we get a further runtime improvementof a factor of $\mathcal{O}(|\Gamma|)$.

For EPDA, our stringsum algorithm is more space-efficient than the algorithm of Alonso et al. (2001) by a factor of $|\Gamma|^2$ and has an improved runtime by a factor of $|\Gamma|^3$, where $|\Gamma|$ is the size of the stack alphabet. Their algorithm is designed for an EPDA without finite-state control, which is equivalent to a PDA ▷ PDA where both the controller and the controllee are single-state. Therefore, we exclude the states when comparing the two algorithms.

5 Computing Allsums

We reuse the notion of pop computation that we defined in §4.1 in order to derive a space-efficient algorithm for computing allsums of CFG ▷ CFG. Allsums of PDA ▷ CFG, CFG ▷ PDA, and PDA ▷ PDA can be computed similarly (see App. D). We define a new type of pop computation that will help us compute the weight of *all* derivations instead of derivations of a particular string.

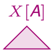
Definition 11. *Let* $\mathbf{G}_1$ *be a controller WCFG and* $\mathbf{G}_2$ *a WLD-CFG, with nonterminal alphabets* $\mathcal{N}_1$ *and* $\mathcal{N}_2$*, respectively. Let be a symbol not occurring in the rules of* $\mathbf{G}_1 \triangleright \mathbf{G}_2$*. A* ***pop computation*** *of type* 

, where $A \in \mathcal{N}_1$ *and* $X, Y \in \mathcal{N}_2$*, is a pair of partial derivations* (d_1, d_2)*, where*

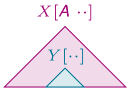
$$d_1 \in (X[A] \overset{*}{\Rightarrow} \boldsymbol{s}_1\, Y[\varepsilon]\boldsymbol{\Upsilon}_2)$$

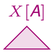
$$d_2 \in (\boldsymbol{\Upsilon}_2 \overset{*}{\Rightarrow} \boldsymbol{s}_2)$$

where $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \Sigma^*$ *and* $\boldsymbol{\Upsilon}_2 \in \mathcal{N}_1[\mathcal{N}_2^*]^*$*. The weight of this pop computation is* $\mathbf{w}(d_1, d_2) = \mathbf{w}(d_1) \otimes \mathbf{w}(d_2)$*.*

A ***pop computation*** *of type* 

is a partial derivation $X[A] \overset{*}{\Rightarrow} \boldsymbol{s}$*, where* $\boldsymbol{s} \in \Sigma^*$*.*

Fig. 4b shows the deduction system for computing allsums. Just as for stringsums, we distinguish two types of items, those of type 

and 

.

Theorem 2. *Let $\mathbf{G}_1$ be a controller WCFG with nonterminal alphabet $\mathcal{N}_1$ and start symbol S, and $\mathbf{G}_2$ a WLD-CFG with nonterminal alphabet $\mathcal{N}_2$. Assume that both have weights in an ω-continuous semiring (App. A.1). Then the allsum of $\mathbf{G}_1 \triangleright \mathbf{G}_2$ is the value of the item $\overset{S[\mathsf{S}]}{\triangle}$ in the deductive system of Fig. 4b.*

Unlike for stringsums, this deduction system has cycles. The value of the goal item $\overset{S[\mathsf{S}]}{\triangle}$ can be found using fixed-point iteration (Goodman, 1999) or the semiring generalization of Newton's method (Esparza et al., 2007).

Since there are various algorithms for computing item values, we don't analyze time complexity, but only space complexity. Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be the sets of nonterminals or stack symbols of the controller and the controllee, respectively. As the algorithm needs to store an item of the form $\overset{X[\mathsf{A}\,\cdot\cdot]}{\underset{Y[\cdot\cdot]}{\triangle}}$ for each $X, Y \in \mathcal{N}_2$ and $\mathsf{A} \in \mathcal{N}_1$, and an item of the form $\overset{X[\mathsf{A}]}{\triangle}$ for each $X \in \mathcal{N}_2$ and $\mathsf{A} \in \mathcal{N}_1$, it has a space complexity of $\mathcal{O}(|\mathcal{N}_1||\mathcal{N}_2|^2)$. If we were to compute allsums in LIG using an algorithm based on Vijay-Shanker and Weir's (1989) algorithm, we would have stored $\mathcal{O}(|\mathcal{N}_1|^2|\mathcal{N}_2|^2)$ items, therefore we have a space improvement of a factor of $|\mathcal{N}_1|$. Similarly, we get a space improvement for EPDA of a factor of $|\mathcal{N}_1|^2$ over the allsum algorithm based on Alonso et al.'s (2001) algorithm.

6 Conclusion

Our work has contributed several new results and algorithms for $\mathcal{L}_2$ formalisms. We introduced semiring-weighted versions of controllable CFGs and PDAs, which give rise naturally to four semiring-weighted two-level formalisms when they are controlled by semiring-weighted CFGs and PDAs. We also introduced a WPDA normal form that is completely analogous to the Chomsky normal form for CFGs and showed that one can derive normal forms for the two-level formalisms only from the normal forms of the controller and of the controllee. These normal forms are also normal forms for TAG, LIG, PAA, and EPDA, respectively, and the conversions only require conversions of the controller and the controllee. Finally, we designed new stringsum and allsum algorithms for all of these formalisms, some of which are faster or more space-efficient than several existing algorithms.

Limitations

Our stringsum algorithms can be used for LIG, EPDA, spinal TAG, and spinal PAA by first converting them to a two-level formalism, converting the resulting controller and controllee grammar/automaton into the normal form, and then merging their rules into a single set of rules. Similarly, for a general two-level formalism, the rules of the controller and the controllee would have to be extracted from the merged rules, converted into the normal form, and merged back before using the stringsum or allsum algorithm. Although simpler, requiring only CFG and PDA conversions, these transformations add some extra complexity. We leave direct normal form conversions for future work.

Ethics Statement

The authors foresee no ethical concerns with the research presented in this paper.

References

Alfred V. Aho and Jeffrey D. Ullman. 1972. *The Theory of Parsing, Translation, and Compiling*, volume 1. Prentice-Hall.

Miguel A. Alonso, Éric Villemonte de la Clergerie, and Manuel Vilares. 2001. A formal definition of bottom-up embedded push-down automata and their tabulation technique. In *Proceedings of the International Conference on Logical Aspects of Computational Linguistics (LACL)*, page 44–61.

Miguel A. Alonso, Éric Villemonte de la Clergerie, and Manuel Vilares. 2000. A redefinition of embedded push-down automata. In *Proceedings of the Fifth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+5)*, pages 19–26.

Y. Bar-Hillel, M. Perles, and E. Shamir. 1961. On formal properties of simple phrase structure grammars. 14:143–172.

Alexandra Butoi, Ryan Cotterell, and David Chiang. 2023. Convergence and diversity in the control hierarchy. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7597–7616. Association for Computational Linguistics.

Alexandra Butoi, Brian DuSell, Tim Vieira, Ryan Cotterell, and David Chiang. 2022. Algorithms for weighted pushdown automata. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Javier Esparza, Stefan Kiefer, and Michael Luttenberger. 2007. On fixed point equations over commutative semirings. In *Proceedings of the Annual Symposium* *on Theoretical Aspects of Computer Science (STACS)*, pages 296–307.

Gerald Gazdar. 1988. Applicability of indexed grammars to natural languages. In *Natural Language Parsing and Linguistic Theories*, pages 69–94.

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

John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. 2006. *Introduction to Automata Theory, Languages, and Computation*, 3rd edition. Addison-Wesley Longman Publishing Co.

Aravind K. Joshi, Leon S. Levy, and Masako Takahashi. 1975. Tree adjunct grammars. *Journal of Computer and System Sciences*, 10(1):136–163.

Aravind K. Joshi, K. Vijay-Shanker, and David J. Weir. 1991. The convergence of mildly context-sensitive grammar formalisms. In *Foundational Issues in Natural Language Processing*, pages 31–81.

Makoto Kanazawa. 2014. A generalization of linear indexed grammars equivalent to simple context-free tree grammars. In *Formal Grammar*, pages 86–103.

Marco Kuhlmann, Carlos Gómez-Rodríguez, and Giorgio Satta. 2011. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 673–682. Association for Computational Linguistics.

Bernard Lang. 1974. Deterministic techniques for efficient non-deterministic parsers. In *ICALP 1974: Automata, Languages and Programming*, pages 255–269.

Harry R. Lewis and Christos H. Papadimitriou. 1997. *Elements of the Theory of Computation*, 2nd edition. Prentice-Hall.

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

K. Vijay-Shanker. 1987. *A Study of Tree Adjoining grammars*. Ph.D. thesis, University of Pennsylvania.

K. Vijay-Shanker and David J. Weir. 1989. Recognition of combinatory categorial grammars and linear indexed grammars. In *Proceedings of the First International Workshop on Parsing Technologies (IWPT)*, pages 172–181.

David J. Weir. 1992. A geometric hierarchy beyond context-free languages. *Theoretical Computer Science*, 104(2):235–261.

A Additional Preliminaries

A.1 Semirings

Definition 12. *A **monoid** is a tuple $(W, \odot, \boldsymbol{I})$, where W is a set, $\odot$ is an associative binary operation, and $\boldsymbol{I} \in W$, called the **identity** element, satisfies $\boldsymbol{I} \odot a = a \odot \boldsymbol{I} = a$ for all $a \in W$. If $a \odot b = b \odot a$ for all a, b, we say that the monoid is **commutative**.*

Definition 13. *A **semiring** is a tuple $\mathcal{W} = (W, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ such that $(W, \oplus, \mathbf{0})$ is a commutative monoid and $(W, \otimes, \mathbf{1})$ is a monoid. Additionally, $\otimes$ distributes over $\oplus$, that is, $a \otimes (b \oplus c) = a \otimes b \oplus a \otimes c$ and $(a \oplus b) \otimes c = a \otimes c \oplus b \otimes c$, and $\mathbf{0}$ is absorbing with respect to $\otimes$, that is, $\mathbf{0} \otimes a = a \otimes \mathbf{0} = \mathbf{0}$. If $\otimes$ is commutative, then we say that $\mathcal{W}$ is **commutative**.*

The following definitions are from Esparza et al. (2007):

Definition 14. *If $\mathcal{W} = (W, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a semiring, the **natural order** on $\mathcal{W}$ is the relation $\leq$, defined such that $a \leq b \Leftrightarrow \exists d \in W : a \oplus d = b$. If $\leq$ is a partial order (i.e., reflexive, antisymmetric, and transitive), then we say that $\mathcal{W}$ is **naturally ordered**.*

Definition 15. *A naturally ordered semiring $\mathcal{W} = (W, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is ω-*continuous *if it is additionally equipped with an infinite summation operator $\bigoplus$ such that:*

- *For any countable sequence $(a_1, a_2, \ldots)$, the sequence $\left(\bigoplus_{i=1}^{k} a_i\right)_{k \geq \mathbf{0}}$ has a least upper bound in W and is equal to $\bigoplus\limits_{i} a_i$.*
- *Multiplication distributes over infinite sums:*

$$\bigoplus_{i}(c \otimes a_i) = c \otimes \left(\bigoplus_{i} a_i\right)$$
$$\bigoplus_{i}(a_i \otimes c) = \left(\bigoplus_{i} a_i\right) \otimes c.$$

- *For any partition $(I_1, I_2, \ldots)$ of $\mathbb{N}$,*

$$\bigoplus_{j} \bigoplus_{i \in I_j} a_i = \bigoplus_{i} a_i.$$

A.2 Weighted Pushdown Automata

We use a definition of WPDA (Butoi et al., 2022) that is more general than usual and roughly a weighted version of the extended PDAs of Aho and Ullman (1972, p. 173) and the PDAs of Lewis and Papadimitriou (1997, p. 131).

Definition 16. *A **weighted pushdown automaton** (WPDA) over a semiring $\mathcal{W} = W$ is a tuple $\mathbf{P} = (Q, \Sigma, \Gamma, \delta, \mathrm{w}, (q_\iota, \boldsymbol{\gamma}_\iota), (q_f, \boldsymbol{\gamma}_f))$, where*

- *Q, Σ, and Γ are finite sets of states, input symbols, and stack symbols, respectively,*
- *$\delta \subseteq Q \times \Gamma^* \times (\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma^*$ is a finite set of transitions,*
- *$\mathrm{w} \colon \delta \to W$ is a transition-weighting function, and*
- *$(q_\iota, \boldsymbol{\gamma}_\iota)$ and $(q_f, \boldsymbol{\gamma}_f)$, where $q_\iota, q_f \in Q, \boldsymbol{\gamma}_\iota, \boldsymbol{\gamma}_f \in \Gamma^*$, are called the initial and final configurations.*

If $(p, \boldsymbol{\gamma}, a, q, \boldsymbol{\gamma}') \in \delta$ is a transition, we write it as $p, \boldsymbol{\gamma} \xrightarrow{a} q, \boldsymbol{\gamma}'$. If $\mathrm{w}(p, \boldsymbol{\gamma} \xrightarrow{a} q, \boldsymbol{\gamma}') = w$, we use the notation $p, \boldsymbol{\gamma} \xrightarrow{a/w} q, \boldsymbol{\gamma}'$.

We treat strings in Γ^* as stacks, with the first symbol being the top of the stack and the last symbol being the bottom.

A WPDA moves from one configuration to another by following transitions of the form $(p, \boldsymbol{\gamma} \xrightarrow{a} q, \boldsymbol{\gamma}')$, which represents a move from state p to state q, while scanning the symbol a, popping the sequence $\boldsymbol{\gamma}$ from the top of the stack and pushing the sequence $\boldsymbol{\gamma}'$. In the following definitions, let $\mathbf{P} = (Q, \Sigma, \Gamma, \delta, \mathrm{w}, (q_\iota, \boldsymbol{\gamma}_\iota), (q_f, \boldsymbol{\gamma}_f))$ be a WPDA.

Definition 17. *A **configuration** of $\mathbf{P}$ is a pair $(q, \boldsymbol{\gamma})$, where $q \in Q$ is the current state and $\boldsymbol{\gamma} \in \Gamma^*$ is the current contents of the stack. We write $\mathcal{C}(\mathbf{P})$ for the set of all configurations of $\mathbf{P}$.*

Definition 18. *If $(p, \boldsymbol{\gamma\alpha})$ and $(q, \boldsymbol{\gamma}'\boldsymbol{\alpha})$ are configurations of $\mathbf{P}$ and $\tau = (p, \boldsymbol{\gamma} \xrightarrow{a} w, q\boldsymbol{\gamma}')$ is a transition of $\mathbf{P}$, we write $(p, \boldsymbol{\gamma\alpha}) \overset{\tau}{\Rightarrow} (q, \boldsymbol{\gamma}'\boldsymbol{\alpha})$ to denote that configuration $(q, \boldsymbol{\gamma}'\boldsymbol{\alpha})$ can be reached from $(p, \boldsymbol{\gamma\alpha})$ in one step using transition τ.*

Definition 19. *We say that a transition τ is k-pop, l-push if $|\boldsymbol{\gamma}| = k$ and $|\boldsymbol{\gamma}'| = l$. We say that τ **scans** a, and if $a \neq \varepsilon$, we call τ **scanning**; otherwise, we call it **non-scanning**.*

The following type of WPDA is a weighted version of the PDA used by Lang (1974) for his recognition algorithm.

Definition 20. *A WPDA is called **simple** if each of its transitions is k-pop, l-push for some $k \leq 1$ and $l \leq 1$.*

The following type of WPDA is a weighted version of the PDA used by (Hopcroft et al., 2006) and others.

Definition 21. *A WPDA is called **top-down** if all of its transitions are 1-pop, and the initial and final configurations are (q_ι, S) and (q_f, ε), respectively.*

Definition 22. *A **run** of $\mathbf{P}$ from (q_0, γ_0) to (q_n, γ_n) is a sequence of steps $\boldsymbol{\pi} = (q_0, \gamma_0) \xRightarrow{\tau_1} \cdots \xRightarrow{\tau_n} (q_n, \gamma_n)$. We write $(q_0, \gamma_0) \overset{*}{\Rightarrow} (q_n, \gamma_n)$ to assert that a run from (p, γ) to (q, γ') exists, or to denote the set of all such runs. If $(q_0, \gamma_0) = (q_\iota, \gamma_\iota)$ and $(q_n, \gamma_n) = (q_f, \gamma_f)$, we call $\boldsymbol{\pi}$ **accepting**. If τ_i scans a_i, for $i \in [1{:}n]$, then we say that $\boldsymbol{\pi}$ scans $\boldsymbol{s} = a_1 \cdots a_n$.*

*The **weight** of $\boldsymbol{\pi}$ is the product of its transition weights,*

$$\mathbf{w}(\boldsymbol{\pi}) \overset{\text{def}}{=} \bigotimes_{i=1}^{n} \mathrm{w}(\tau_i).$$

We denote by $\Pi(\mathbf{P}, \boldsymbol{s})$ the set all accepting runs of $\mathbf{P}$ scanning $\boldsymbol{s}$, and by $\Pi(\mathbf{P})$ the set of all accepting runs of $\mathbf{P}$.

Definition 23. *The **stringsum** $\mathbf{w}(\mathbf{P}, \boldsymbol{s})$ of a string $\boldsymbol{s}$ under $\mathbf{P}$ is the total weight of all accepting runs scanning $\boldsymbol{s}$,*

$$\mathbf{w}(\mathbf{P}, \boldsymbol{s}) \overset{\text{def}}{=} \bigoplus_{\boldsymbol{\pi} \in \Pi(\mathbf{P}, \boldsymbol{s})} \mathbf{w}(\boldsymbol{\pi}).$$

Definition 24. *The **allsum** $\mathbf{w}(\mathbf{P})$ of $\mathbf{P}$ is the total weight of all its accepting runs,*

$$\mathbf{w}(\mathbf{P}) \overset{\text{def}}{=} \bigoplus_{\boldsymbol{\pi} \in \Pi(\mathbf{P})} \mathbf{w}(\boldsymbol{\pi}).$$

B Two-Level Formal Systems

B.1 Definitions

Having defined both WPDA (Def. 16) and WLD-CFG (Def. 7), we can give a definition for weighted PDA ▷ CFG.

Definition 25. *Let $\mathbf{P}_1$ be a controller WPDA with states Q, stack alphabet Γ_1 and initial configuration (q_ι, γ_ι), and let $\mathbf{G}_2$ be a controllee WLD-CFG with nonterminals $\mathcal{N}_2$. Then $\mathbf{P}_1 \triangleright \mathbf{G}_2$ is a rewriting system with rules as follows.*

(a) If $(q, \mathsf{A} \xrightarrow{\varepsilon/w} r, \boldsymbol{\gamma})$ is a transition of $\mathbf{P}_1$, then $\mathbf{P}_1 \triangleright \mathbf{G}_2$ has a rule for each $X \in \mathcal{N}_2$:

$$X[q, \mathsf{A} \cdot\cdot] \xrightarrow{w} X[r, \boldsymbol{\gamma} \cdot\cdot].$$

(b) If $(r, \mathsf{A} \xrightarrow{\ell/w_1} s, \varepsilon)$ is a transition of $\mathbf{P}_1$, and $(\ell \colon X \xrightarrow{w_2} \boldsymbol{\alpha}_1 \breve{Y} \boldsymbol{\alpha}_2)$ is a production of $\mathbf{G}_2$, then $\mathbf{G}_1 \triangleright \mathbf{G}_2$ has a rule

$$X[r, \mathsf{A} \cdot\cdot] \xrightarrow{w_1 \otimes w_2} \boldsymbol{\alpha}_1[q_\iota, \boldsymbol{\gamma}_\iota]\ Y[s, \cdot\cdot]\ \boldsymbol{\alpha}_2[q_\iota, \boldsymbol{\gamma}_\iota].$$

Next, we define WLD-PDAs by analogy with WLD-CFGs. This definition is a weighted version of our previous definition (Butoi et al., 2023).

Definition 26. *A **weighted labeled distinguished pushdown automaton** (WLD-PDA) over a semiring $\mathcal{W} = (W, \oplus, \otimes, \mathbf{0}, \mathbf{1})$ is a tuple $\mathbf{P} = (Q, \Sigma, \Gamma, L, \delta, \mathrm{w}, (q_\iota, S), (q_f, \varepsilon))$, where*

- *Q, Σ, Γ and L are finite sets of states, input symbols, stack symbols and labels, respectively,*
- *$\delta \subseteq L \times \mathbb{N} \times Q \times \Gamma \times (\Sigma \cup \{\varepsilon\}) \times Q \times \Gamma^*$ is a finite set of transitions,*
- *$\mathrm{w} \colon \delta \to W$ is the transition-weighting function, and*
- *(q_ι, S) and (q_f, ε), where $q_\iota, q_f \in Q$, $S \in \Gamma$, are the initial and final configurations.*

If $(\ell, i, q, \mathsf{A}, a, r, \mathsf{B}_1 \cdots \mathsf{B}_n)$ is a transition in δ, we must have either $i = 0$, which we write as $q, \mathsf{A} \xrightarrow{a} r, \mathsf{B}_1 \cdots \mathsf{B}_n$, or $1 \le i \le n$, which we write as $q, \mathsf{A} \xrightarrow{a} r, \mathsf{B}_1 \cdots \mathsf{B}_{i-1} \breve{\mathsf{B}}_i \mathsf{B}_{i+1} \cdots \mathsf{B}_n$.

A WLD-PDA behaves similarly to a WPDA, but its runs are controlled by either a controller CFG or PDA. As in §3, the rules of the controller and of the controllee can be merged into a single system.

Definition 27. *Let $\mathbf{G}_1$ be a controller CFG with nonterminals $\mathcal{N}$ and start symbol S, and let $\mathbf{P}_2$ be a WLD-PDA with stack alphabet Γ and states Q. Then $\mathbf{G}_1 \triangleright \mathbf{P}_2$ is a rewriting system with rules as follows.*

(a) If $(\mathsf{A} \xrightarrow{w} \boldsymbol{\beta})$ is a production of $\mathbf{G}_1$, where $\boldsymbol{\beta} \in \mathcal{N}^*$, then $\mathbf{G} \triangleright \mathbf{P}_2$ has a transition for each $X \in \Gamma$ and $q \in Q$:

$$q, X[\mathsf{A} \cdot\cdot] \xrightarrow{\varepsilon/w} q, X[\boldsymbol{\beta} \cdot\cdot].$$

(b) If $(\mathsf{A} \xrightarrow{w_1} \ell)$ is a production of $\mathbf{G}_1$, and $\ell \colon q, X \xrightarrow{a/w_2} r, \boldsymbol{\alpha}_1 \breve{Y} \boldsymbol{\alpha}_2$ is a transition of $\mathbf{P}_2$, then $\mathbf{G} \triangleright \mathbf{P}_2$ has a transition

$$q, X[\mathsf{A} \cdot\cdot] \xrightarrow{a/w_1 \otimes w_2} r, \boldsymbol{\alpha}_1[\mathsf{S}]\ Y[\cdot\cdot]\ \boldsymbol{\alpha}_2[\mathsf{S}].$$

Definition 28. *Let $\mathbf{P}_1$ be a controller PDA with stack alphabet Γ_1 and start configuration $(q_\iota, \boldsymbol{\gamma}_\iota)$, and let $\mathbf{P}_2$ be a WLD-PDA with stack alphabet Γ_2 and states Q_2. Then $\mathbf{P}_1 \triangleright \mathbf{P}_2$ is a rewriting system with rules as follows.*

(a) If $(q, \mathsf{A} \xrightarrow{\varepsilon/w} r, \boldsymbol{\gamma})$ is a transition of $\mathbf{P}_1$, then $\mathbf{P}_1 \triangleright \mathbf{P}_2$ has a transition for each $X \in \mathcal{N}_2$ and $p \in Q_2$:

$$p, X[q, \mathsf{A} \cdot\cdot] \xrightarrow{\varepsilon/w} p, X[r, \boldsymbol{\gamma} \cdot\cdot].$$

(b) If $(r, \mathsf{A} \xrightarrow{\ell/w_1} s, \varepsilon)$ is a transition of $\mathbf{P}_1$, and $\ell \colon p, X \xrightarrow{a/w_2} q, \boldsymbol{\alpha}_1 \breve{Y} \boldsymbol{\alpha}_2$ is a transition of $\mathbf{P}_2$, then $\mathbf{G} \triangleright \mathbf{P}_2$ has a transition

$$p, X[r, \mathsf{A} \cdot\cdot] \xrightarrow{a/w_1 \otimes w_2} q, \boldsymbol{\alpha}_1[q_\iota, \boldsymbol{\gamma}_\iota]\, Y[s, \cdot\cdot] \boldsymbol{\alpha}_2[q_\iota, \boldsymbol{\gamma}_\iota].$$

The terms scanning/non-scanning transitions and (accepting) runs are defined analogously to those for WPDAs.

B.2 Normal Forms

B.2.1 WCFG normal form

Proposition 1. *For any WCFG $\mathbf{G}_1$ with weights in an ω-continuous semiring, there is a WCFG in Chomsky normal form that defines the same weighted language as $\mathbf{G}_1$.*

For any WLD-CFG $\mathbf{G}_2$ with weights in an ω-continuous semiring, there is a WLD-CFG in Chomsky normal form that is equivalent to $\mathbf{G}_2$.

Proof. The proof for WCFGs is a generalization of the standard conversion for unweighted grammars (Hopcroft et al., 2006), and is related to the Earley-style algorithm of Stolcke (1995).

1. If S is the old start symbol, add a new nonterminal S', a production $S' \xrightarrow{\mathbf{1}} S$, and make S' the new start symbol.

2. For every terminal a, create a production $T_a \xrightarrow{\mathbf{1}} a$, where T_a is a fresh nonterminal, and replace every right-hand side occurrence of a with T_a.

3. Replace every production $X \xrightarrow{w} Y_1 \cdots Y_k$, where $k > 2$, with the productions

$$\begin{aligned} X &\xrightarrow{w} Y_1 Z_1 \\ Z_1 &\xrightarrow{\mathbf{1}} Y_2 Z_2 \\ &\vdots \\ Z_{k-2} &\xrightarrow{\mathbf{1}} Y_{k-1} Y_k \end{aligned}$$

where $Z_1, \ldots, Z_{k-2}$ are fresh nonterminals.

4. For each nonterminal X, compute the weight of all partial derivations $X \overset{*}{\Rightarrow} \varepsilon$ using the deductive system

$$\frac{}{A \overset{*}{\Rightarrow} \varepsilon} \quad A \xrightarrow{w} \varepsilon$$

$$\frac{B \overset{*}{\Rightarrow} \varepsilon}{A \overset{*}{\Rightarrow} \varepsilon} \quad A \xrightarrow{w} B$$

$$\frac{B \overset{*}{\Rightarrow} \varepsilon \quad C \overset{*}{\Rightarrow} \varepsilon}{A \overset{*}{\Rightarrow} \varepsilon} \quad A \xrightarrow{w} BC$$

Then for every production $X \xrightarrow{w} YZ$ with $\mathbf{w}(Y \overset{*}{\Rightarrow} \varepsilon) = w_Y$ and $\mathbf{w}(Z \overset{*}{\Rightarrow} \varepsilon) = w_Z$, add productions $X \xrightarrow{w_1 \otimes w_Y} Z$ and $X \xrightarrow{w_1 \otimes w_Z} Y$. Finally, for all X, remove all productions $X \rightarrow \varepsilon$.

5. For all nonterminals X and Y, compute the weight of all partial derivations $X \overset{*}{\Rightarrow} Y$ using the deductive system

$$\frac{}{A \overset{*}{\Rightarrow} A}$$

$$\frac{A \overset{*}{\Rightarrow} B}{A \overset{*}{\Rightarrow} C} \quad B \xrightarrow{w} C$$

Then for every production $X \xrightarrow{w_2} YZ$ and nonterminal $W \neq X$, with $\mathbf{w}(W \overset{*}{\Rightarrow} X) = w_1$, add production $W \xrightarrow{w_1 \otimes w_2} YZ$. Finally, for all X, Y, remove all productions $X \rightarrow Y$.

Now we give a similar construction for WLD-CFGs. Let $\mathbf{G}$ be a WLD-CFG and $\mathbf{F}$ a controller WCFG or WPDA, both with weights from an ω-continuous semiring. Then there exists a WLD-CFG in normal form $\mathbf{G}'$ and a controller $\mathbf{F}'$ such that $\mathbf{F}' \triangleright \mathbf{G}'$ defines the same weighted language as $\mathbf{F} \triangleright \mathbf{G}$. We assume for simplicity that $\mathbf{F}$ is a controller WCFG; a similar construction can be given when it is a WPDA.

Converting a CFG ▷ CFG to normal form requires an allsum algorithm that works on any binarized CFG ▷ CFG, not necessarily in normal form. We must use the rules shown in Fig. 5 in addition to those from Fig. 4b when computing allsums.

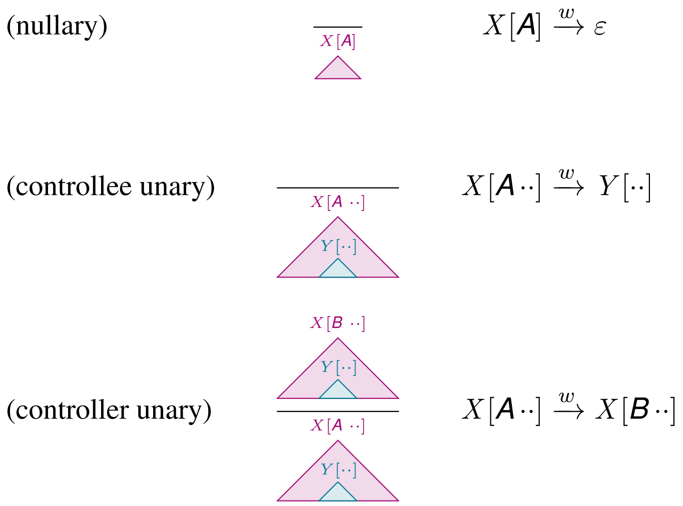

(nullary)
$X[\mathsf{A}]$
$X[\mathsf{A}] \xrightarrow{w} \varepsilon$
(controllee unary)
$X[\mathsf{A}\cdot\cdot]$
$Y[\cdot\cdot]$
$X[\mathsf{A}\cdot\cdot] \xrightarrow{w} Y[\cdot\cdot]$
(controller unary)
$X[\mathsf{B}\cdot\cdot]$
$Y[\cdot\cdot]$
$X[\mathsf{A}\cdot\cdot]$
$Y[\cdot\cdot]$
$X[\mathsf{A}\cdot\cdot] \xrightarrow{w} X[\mathsf{B}\cdot\cdot]$

Figure 5: Additional deduction rules for computing allsums in binarized CFG ▷ CFG.

Root–foot transform The nullary and unary removal steps below are analogous to nullary and unary removal in a WCFG. However, sometimes, when we remove nullary/unary rules, we need information from both the controller and controllee to compute their lost weight. To facilitate this, we define the following transformation, which copies information from the controllee to the controller.

Let $\mathcal{N}_1$ and $\mathcal{N}_2$ be the controller and controllee nonterminal alphabet, respectively.

1. For every controllee rule $\ell : X \to \boldsymbol{\alpha}$ where $\boldsymbol{\alpha}$ has no distinguished symbol, relabel the rule as $\ell^X_\perp$.

2. For every controllee rule $\ell : X \to \boldsymbol{\alpha}\, \overset{\smile}{Y} \boldsymbol{\beta}$, relabel the rule as ℓ^X_Y.

3. Perform the construction of Bar-Hillel et al. (1961) on the controller, using controllee nonterminals instead of states. That is, relabel every controller nonterminal A as $\mathsf{A}^X_\perp$ or A^X_Y for all $X, Y \in \mathcal{N}_2$, making as many copies of rules as needed, subject to the constraints that (i) if the lhs is A^X_Y (where Y could be $\perp$) then the first rhs symbol must be $\mathsf{A}^X_{Y'}$, and the last rhs symbol must be $\mathsf{A}^{X'}_Y$; (ii) if A^X_Y is an rhs nonterminal then its successor (if any) must be B^Y_Z.

4. Add a new controller start symbol S' with rules $\mathsf{S}' \to \mathsf{S}^X_\perp$ for all $X \in \mathcal{N}_2$.

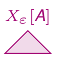
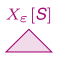
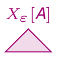
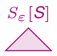
Controllee binarization In the controllee, replace every production $\ell : X \xrightarrow{w} Y_1 \cdots \overset{\smile}{Y_d} \cdots Y_k$, where $k > 2$, with the productions

$$\begin{aligned}
\ell_1 : X &\xrightarrow{w} Y_1 \overset{\smile}{Z_1} \\
\ell_2 : Z_1 &\xrightarrow{\mathbf{1}} Y_2 \overset{\smile}{Z_2} \\
&\vdots \\
\ell_{d-1} : Z_{d-2} &\xrightarrow{\mathbf{1}} Y_{d-1} \overset{\smile}{Z_d} \\
\ell_d : Z_d &\xrightarrow{\mathbf{1}} \overset{\frown}{Z_{d+1}}\, Y_k \\
\ell_{d+1} : Z_{d+1} &\xrightarrow{\mathbf{1}} \overset{\frown}{Z_{d+2}}\, Y_{k-1} \\
&\vdots \\
\ell_{k-1} : Z_{k-1} &\xrightarrow{\mathbf{1}} \overset{\smile}{Y_d}\, Y_{d+1}
\end{aligned}$$

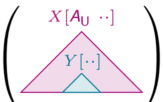
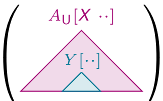
where $Z_1, \ldots, Z_{k-2}$ are fresh controllee nonterminals.

Accordingly, in the controller, replace every rhs occurrence of ℓ with $\ell_1 \cdots \ell_{k-1}$.

Controllee nullary removal Like the standard nullary removal algorithm (Hopcroft et al., 2006), nullary removal in an WLD-CFG has three steps: partitioning, precomputation, and removal. However, removing nullary rules from the controllee also sometimes requires modifications to the controller.

1. Partition: Replace every controllee nonterminal X with two nonterminals X_ε and $X_{\not\varepsilon}$. Every rule with k rhs nonterminals becomes 2^k rules with all possible combination of rhs nonterminals, and the lhs annotated ε iff the rhs is empty or all the rhs symbols are annotated ε. The start symbol is $S_{\not\varepsilon}$.
2. Precompute: Compute the allsum $\mathbf{w}$ () for all $\mathsf{A} \in \mathcal{N}_1, X_\varepsilon \in \mathcal{N}_2$.
3. Remove (non-distinguished): For every non-distinguished occurrence of X_ε in the rhs of a controllee rule π, delete the occurrence and multiply the weight of π by $\mathbf{w}$ ().
4. Remove (distinguished): The distinguished occurrences of X_ε cannot be removed in the same way, because we don't know what weight to multiply in. Instead, perform the root–foot transform. Then, for every occurrence of $\mathsf{A}_\perp^{X_\varepsilon}$ on the rhs of a controller rule π, delete the occurrence and multiply the weight of π by $\mathbf{w}$ ().
5. Remove (start): Add controller rule $\mathsf{S}' \xrightarrow{\mathbf{1}} \ell$ and controllee rule $\ell : S^{\not\varepsilon} \xrightarrow{w} \varepsilon$, where $w = \mathbf{w}$ ().
6. Remove every controller rule $\mathsf{S}' \to \varepsilon$.
7. Remove every controller rule with lhs $\mathsf{A}_\perp^{X_\varepsilon}$ and every controllee rule with lhs X_ε.

Controllee unary removal

1. Partition: Replace every controller nonterminal A with two nonterminals A_U and $\mathsf{A}_{\not\mathsf{U}}$. Every rule with k rhs nonterminals becomes 2^k rules. A rule's lhs is annotated U iff all of its rhs symbols are either labels of controllee unary rules or annotated U. The controller temporarily has two start symbols, S_U and $\mathsf{S}_{\not\mathsf{U}}$.
2. Precompute: Compute the allsum $\mathbf{w}$ () for all $A_\mathsf{U} \in \mathcal{N}_1, X, Y \in \mathcal{N}_2$.
3. Remove (controller): Perform the root–foot transform. Then, for every occurrence of $\mathsf{A}_\mathsf{U}{}^{X}_{Y}$ on the rhs of rule π, delete the occurrence and multiply the weight of π by $\mathbf{w}$ (). Remove all rules with lhs $X_Y^{A_\mathsf{U}}$.
4. Remove (controllee): Remove all unary rules, and for each rule $\ell : Y \xrightarrow{w} \boldsymbol{\alpha}$, and all $X \in \mathcal{N}_2$, make a copy $\ell : X \xrightarrow{w} \boldsymbol{\alpha}$.

□

B.2.2 WPDA normal form

We first define a WPDA normal form that is analogous to Chomsky normal form for WCFGs.

Definition 29. *A WPDA is in* ***normal form*** *if all of its transitions are of one of the following types: (1) $q_\iota, S \xrightarrow{\varepsilon/w} q_f, \varepsilon$, (2) scanning and 0-push, or (3) non-scanning and 2-push. Moreover, S is not pushed by any transition.*

A WLD-PDA is in normal form if all of its transitions are of type (1) or (2) above, (3a) $q, X \xrightarrow{\varepsilon/w} r, \breve{Y} Z$, or (3b) $q, X \xrightarrow{\varepsilon/w} r, Y \breve{Z}$.

Proposition 2. *Let $\mathbf{P}$ be a WPDA with weights from an ω-continuous semiring. Then there is a WPDA in normal form that defines the same weighted language as $\mathbf{P}$.*

For any WLD-PDA $\mathbf{P}_2$ with weights in an ω-continuous semiring, there is a WLD-PDA in Chomsky normal form that is equivalent to $\mathbf{P}_2$.

Proof. In previous work (Butoi et al., 2022), we gave a conversion from arbitrary WPDAs to a normal form that is close to Def. 29, but allows transitions that are scanning but are 1-push or 2-push. We can modify the conversion using the following preparatory steps:

1. If (q_ι, γ_ι) is the old start configuration, add a new state $q_\iota{}'$, a new stack symbol S', and a transition $q_\iota{}', S' \xrightarrow{\mathbf{1}} q_\iota, \gamma_\iota$, and make $(q_\iota{}', S')$ the new start configuration.
2. For every terminal a, add a fresh stack symbol T_a. Replace every scanning transition $(q, X \xrightarrow{a/w} r, \gamma)$ with $(q, X \xrightarrow{\epsilon/w} r, T_a\gamma)$, and add transitions $(q, T_a \xrightarrow{a/\mathbf{1}} q, \varepsilon)$ for every state q.
3. Remove all nullary transitions (Butoi et al., 2022, §3.2).
4. Remove all unary transitions (Butoi et al., 2022, §3.3).

A WLD-PDA can always be converted into an equivalent WLD-CFG (Butoi et al., 2023), converted into the normal form, then converted back into into a WLD-PDA. □

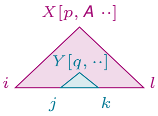
To convert a WLD-PDA to normal form, we use the construction above, modified by analogy with Prop. 1. In the root–foot transform, each nonterminal includes two or four states (just as allsum items do). The nullary removal procedure for WPDAs is more complex than for WCFGs, but the required modification is analogous: we remove non-distinguished stack symbols X_ε as usual, but when removing a distinguished stack symbol X_ε, we must multiply its weight in the controller rather than the controllee.

B.2.3 Two-level normal forms

Fig. 6 shows how the normal forms of PDA ▷ CFG, CFG ▷ PDA and PDA ▷ PDA are obtained from the normal forms of the controllees and the controllers. For the epsilon rules, as noted in §3.2, in principle there could be rules with other left-hand sides, but such rules would never be used.

C Computing Stringsums

C.1 Pop Computations

Definition 30. *Let $\mathbf{P}$ be a controller WPDA with states Q, and stack alphabet Γ, and let $\mathbf{G}$ be a WLD-CFG with nonterminals $\mathcal{N}$. Let $\boldsymbol{s} \in \Sigma^*$ be a string of length $|\boldsymbol{s}| = n$. A* ***pop computation*** *of $\mathbf{P} \triangleright \mathbf{G}$ of type* 

, where $0 \le i \le j \le k \le l \le n$, $p, q \in Q$, $X, Y \in \mathcal{N}$, and $A \in \Gamma$, is a pair of partial derivations (d_1, d_2), where

$$d_1 \in (X[p, A] \overset{*}{\Rightarrow} \boldsymbol{s}_{(i:j)} Y[q, \varepsilon] \boldsymbol{\Upsilon}_2)$$

$$d_2 \in (\boldsymbol{\Upsilon}_2 \overset{*}{\Rightarrow} \boldsymbol{s}_{(k:l)}).$$

$\mathbf{P}_1$	$\mathbf{G}_2$	$\mathbf{P}_1 \triangleright \mathbf{G}_2$	name
$q_\iota, \mathsf{S} \xrightarrow{\ell/w_1} q_f, \varepsilon$	$\ell\colon S \xrightarrow{w_2} \varepsilon$	$S[q_\iota, \mathsf{S}] \xrightarrow{w_1 \otimes w_2} \varepsilon$	(epsilon)
$p, \mathsf{A} \xrightarrow{\ell/w_1} q_f, \varepsilon$	$\ell\colon X \xrightarrow{w_2} a$	$X[p, \mathsf{A}] \xrightarrow{w_1 \otimes w_2} a$	(terminal)
$p, \mathsf{A} \xrightarrow{\ell/w_1} q, \varepsilon$	$\ell\colon X \xrightarrow{w_2} \breve{Y} Z$	$X[p, \mathsf{A} \cdot\cdot] \xrightarrow{w_1 \otimes w_2} Y[q, \cdot\cdot] Z[q_\iota, \mathsf{S}]$	(left pop)
$p, \mathsf{A} \xrightarrow{\ell/w_1} q, \varepsilon$	$\ell\colon X \xrightarrow{w_2} Y \breve{Z}$	$X[p, \mathsf{A} \cdot\cdot] \xrightarrow{w_1 \otimes w_2} Y[q_\iota, \mathsf{S}] Z[q, \cdot\cdot]$	(right pop)
$p, \mathsf{A} \xrightarrow{\varepsilon/w_1} q, \mathsf{BC}$		$X[q, \mathsf{A} \cdot\cdot] \xrightarrow{w_1} X[q, \mathsf{BC} \cdot\cdot]$	(push)

(a) PDA ▷ CFG.

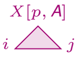
$\mathbf{G}_1$	$\mathbf{P}_2$	$\mathbf{G}_1 \triangleright \mathbf{P}_2$	name
$\mathsf{S} \xrightarrow{w_1} \ell$	$\ell\colon q_\iota, S \xrightarrow{\varepsilon/w_2} q_f, \varepsilon$	$q_\iota, S[\mathsf{S}] \xrightarrow{\varepsilon/w_1 \otimes w_2} q_f, \varepsilon$	(epsilon)
$\mathsf{A} \xrightarrow{w_1} \ell$	$\ell\colon q, X \xrightarrow{a/w_2} r, \varepsilon$	$q, X[\mathsf{A}] \xrightarrow{a/w_1 \otimes w_2} r, \varepsilon$	(terminal)
$\mathsf{A} \xrightarrow{w_1} \ell$	$\ell\colon q, X \xrightarrow{\varepsilon/w_2} r, \breve{Y} Z$	$q, X[\mathsf{A} \cdot\cdot] \xrightarrow{\varepsilon/w_1 \otimes w_2} r,\ Y[\cdot\cdot] Z[\mathsf{S}]$	(left pop)
$\mathsf{A} \xrightarrow{w_1} \ell$	$\ell\colon q, X \xrightarrow{\varepsilon/w_2} r, Y \breve{Z}$	$q, X[\mathsf{A} \cdot\cdot] \xrightarrow{\varepsilon/w_1 \otimes w_2} r,\ Y[\mathsf{S}] Z[\cdot\cdot]$	(right pop)
$\mathsf{A} \xrightarrow{w_1} \mathsf{BC}$		$q, X[\mathsf{A} \cdot\cdot] \xrightarrow{\varepsilon/w_1} q, X[\mathsf{BC} \cdot\cdot]$	(push)

(b) CFG ▷ PDA.

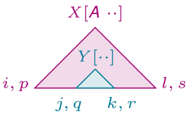
$\mathbf{P}_1$	$\mathbf{P}_2$	$\mathbf{P}_1 \triangleright \mathbf{P}_2$	name
$q^1_\iota, \mathsf{S} \xrightarrow{\ell/w_1} q^1_f, \varepsilon$	$\ell\colon q^2_\iota, S \xrightarrow{\varepsilon/w_2} q^2_f, \varepsilon$	$q^2_\iota, S[q^1_\iota, \mathsf{S}] \xrightarrow{\varepsilon/w_1 \otimes w_2} q^2_f, \varepsilon$	(epsilon)
$p, \mathsf{A} \xrightarrow{\ell/w_1} q^1_f, \varepsilon$	$\ell\colon r, X \xrightarrow{a/w_2} s, \varepsilon$	$r, X[p, \mathsf{A}] \xrightarrow{a/w_1 \otimes w_2} s, \varepsilon$	(terminal)
$p, \mathsf{A} \xrightarrow{\ell/w_1} q, \varepsilon$	$\ell\colon r, X \xrightarrow{\varepsilon/w_2} s, \breve{Y} Z$	$r, X[p, \mathsf{A} \cdot\cdot] \xrightarrow{\varepsilon/w_1 \otimes w_2} s,\ Y[q, \cdot\cdot] Z[q^1_\iota, \mathsf{S}]$	(left pop)
$p, \mathsf{A} \xrightarrow{\ell/w_1} q, \varepsilon$	$\ell\colon r, X \xrightarrow{\varepsilon/w_2} s, Y \breve{Z}$	$r, X[p, \mathsf{A} \cdot\cdot] \xrightarrow{\varepsilon/w_1 \otimes w_2} s,\ Y[q^1_\iota, \mathsf{S}] Z[q, \cdot\cdot]$	(right pop)
$p, \mathsf{A} \xrightarrow{\varepsilon/w_1} q, \mathsf{BC}$		$r, X[p, \mathsf{A} \cdot\cdot] \xrightarrow{\varepsilon/w_1} r, X[q, \mathsf{BC} \cdot\cdot]$	(push)

(c) PDA ▷ PDA. States q^1_ι and q^2_ι are the initial states of $\mathbf{P}_1$ and $\mathbf{P}_2$, respectively, and similarly for q^1_f and q^2_f.

Figure 6: Normal forms of PDA ▷ CFG, CFG ▷ PDA, PDA ▷ PDA resulting from normal forms of their controllers and controllees.

A ***pop computation*** *of type* 

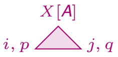
is a partial derivation $X[p, A] \stackrel{*}{\Rightarrow} \boldsymbol{s}_{(i:j]}$.

Definition 31. *Let* $\mathbf{G}$ *be a controller WCFG with nonterminals* $\mathcal{N}$*, and let* $\mathbf{P}$ *be a WLD-PDA with states* Q *and stack alphabet* Γ*. Let* $\boldsymbol{s} \in \Sigma^*$ *be an input string of length* $|\boldsymbol{s}| = n$*. A* ***pop computation*** *of* $\mathbf{G} \triangleright \mathbf{P}$ *of type* 

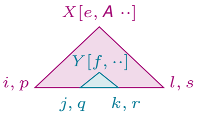
, where $0 \leq i \leq j \leq k \leq l \leq n$*,* $p, q, r, s \in Q$*,* $X, Y \in \Gamma$*, and* $A \in \mathcal{N}$*, is a pair of runs* $(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$*, where*

$$\boldsymbol{\pi}_1 \in ((p, X[A]) \stackrel{*}{\Rightarrow} (q, Y[\varepsilon]\boldsymbol{\Upsilon}_2)) \text{ and scans } \boldsymbol{s}_{(i:j]}$$

$$\boldsymbol{\pi}_2 \in ((r, \boldsymbol{\Upsilon}_2) \stackrel{*}{\Rightarrow} (s, \varepsilon)) \text{ and scans } \boldsymbol{s}_{(k:l]}.$$

A ***pop computation*** *of type* 

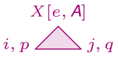
is a run $(p, X[A]) \stackrel{*}{\Rightarrow} (q, \varepsilon)$ *scanning* $\boldsymbol{s}_{(i:j]}$.

Definition 32. *Let* $\mathbf{P}_1$ *be a controller WPDA and* $\mathbf{P}_2$ *be a WLD-PDA, with states and stack alphabets* Q_1 *and* Γ_1*, and* Q_2 *and* Γ_2*, respectively. Let* $\boldsymbol{s} \in \Sigma^*$ *be an input string of length* $|\boldsymbol{s}| = n$*. A* ***pop computation*** *of* $\mathbf{P}_1 \triangleright \mathbf{P}_2$ *of type* 

, where $0 \leq i \leq j \leq k \leq l \leq n$*,* $p, q, r, s \in Q_2$*,* $e, f \in Q_1$*,* $X, Y \in \Gamma_2$*, and* $A \in \Gamma_1$*, is a pair of runs* $(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$*, where*

$$\boldsymbol{\pi}_1 \in ((p, X[e, A]) \stackrel{*}{\Rightarrow} (q, Y[f, \varepsilon]\boldsymbol{\Upsilon}_2)) \text{ and scans } \boldsymbol{s}_{(i:j]}$$

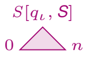
$$\boldsymbol{\pi}_2 \in ((r, \boldsymbol{\Upsilon}_2) \stackrel{*}{\Rightarrow} (s, \varepsilon)) \text{ and scans } \boldsymbol{s}_{(k:l]}.$$

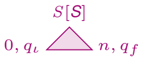
A ***pop computation*** *of type* 

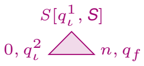
is a run $(p, X[e, A]) \stackrel{*}{\Rightarrow} (q, \varepsilon)$ *scanning* $\boldsymbol{s}_{(i:j]}$.

C.2 Deduction Systems

Figures 7 to 9, in the columns labeled "stringsum," show deduction systems for computing stringsums of PDA ▷ CFG, CFG ▷ PDA, and PDA ▷ PDA. The controller and controllee PDAs are assumed to be single-state.

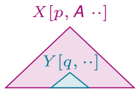
The goal items for PDA ▷ CFG, CFG ▷ PDA, and PDA ▷ PDA are, in order, 

, 

, and 

, where n is the length of the input string.

D Computing Allsums

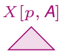
Definition 33. *Let* $\mathbf{P}$ *be a controller WPDA with states* Q *and stack alphabet* Γ*, and let* $\mathbf{G}$ *be a WLD-CFG with nonterminal alphabet* $\mathcal{N}$*. A* ***pop computation*** *of* $\mathbf{P} \triangleright \mathbf{G}$ *of type* 

, where $X, Y \in \mathcal{N}$*,* $p, q \in Q$*, and* $A \in \Gamma$*, is a pair of partial derivations* (d_1, d_2)*, where*

$$d_1 \in (X[p, A] \stackrel{*}{\Rightarrow} \boldsymbol{s}_1 Y[q, \varepsilon]\boldsymbol{\Upsilon}_2)$$

$$d_2 \in (\boldsymbol{\Upsilon}_2 \stackrel{*}{\Rightarrow} \boldsymbol{s}_2)$$

and $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \Sigma^*$*,* $\boldsymbol{\Upsilon}_2 \in \mathcal{N}[\Gamma^*]^*$*. The weight of this pop computation is* $\mathbf{w}(d_1, d_2) = \mathbf{w}(d_1) \otimes \mathbf{w}(d_2)$*.*

A ***pop computation*** *of type* 

is a partial derivation $X[p, A] \stackrel{*}{\Rightarrow} \boldsymbol{s}$*, where* $\boldsymbol{s} \in \Sigma^*$*.*

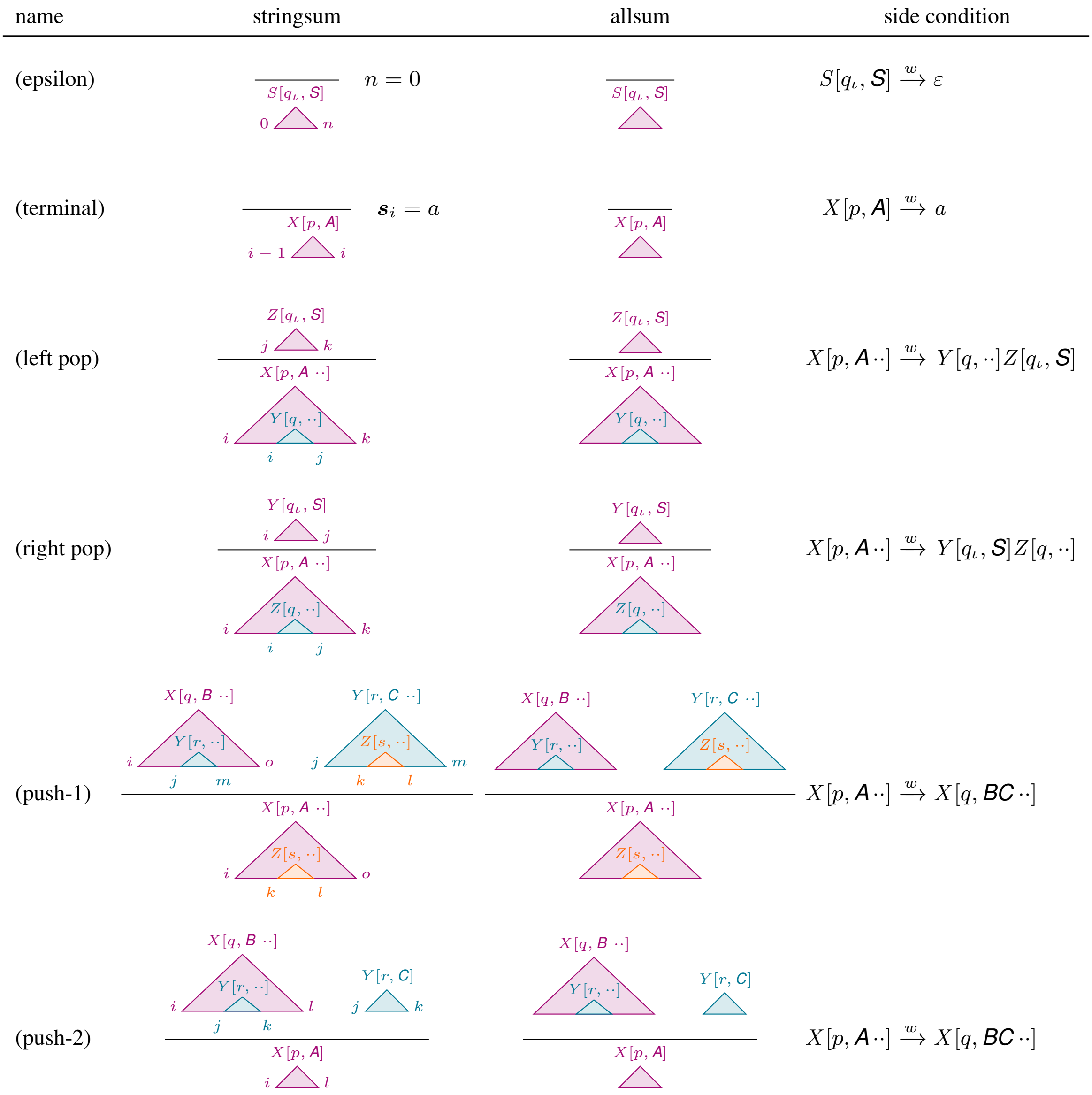


Figure 7: Deductive systems for computing stringsums and allsums of PDA ▷ CFG. State q_ι is the initial state of the controller.

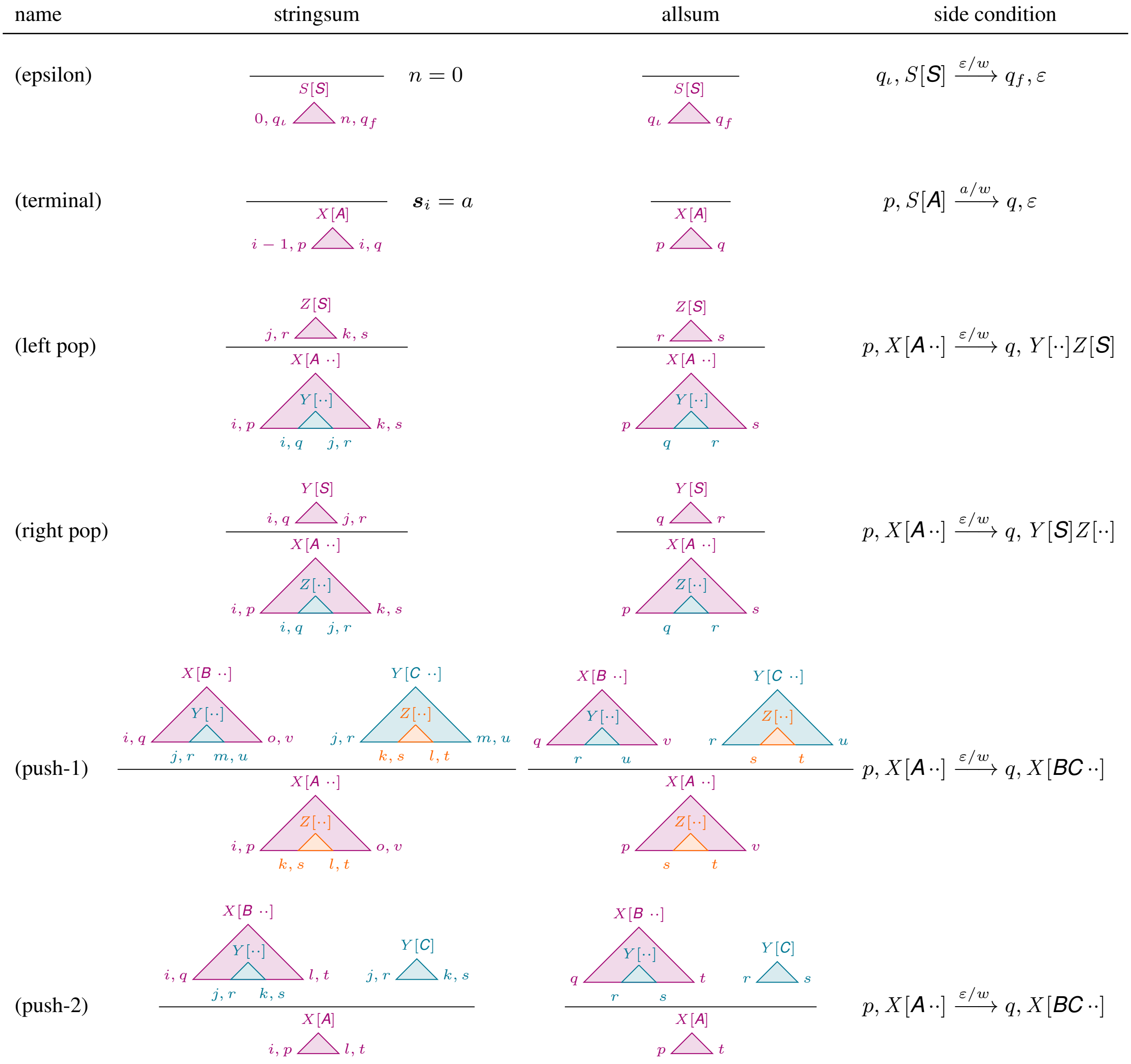

name
stringsum
allsum
side condition
(epsilon)
$n = 0$
$q_\iota, S[S] \xrightarrow{\varepsilon/w} q_f, \varepsilon$
(terminal)
$\boldsymbol{s}_i = a$
$p, S[A] \xrightarrow{a/w} q, \varepsilon$
(left pop)
$p, X[A\,\cdot\cdot] \xrightarrow{\varepsilon/w} q, Y[\cdot\cdot]Z[S]$
(right pop)
$p, X[A\,\cdot\cdot] \xrightarrow{\varepsilon/w} q, Y[S]Z[\cdot\cdot]$
(push-1)
$p, X[A\,\cdot\cdot] \xrightarrow{\varepsilon/w} q, X[BC\,\cdot\cdot]$
(push-2)
$p, X[A\,\cdot\cdot] \xrightarrow{\varepsilon/w} q, X[BC\,\cdot\cdot]$

Figure 8: Deductive systems for computing stringsums and allsums of CFG ▷ PDA. States q_ι and q_f are the controllee's initial and final states, respectively.

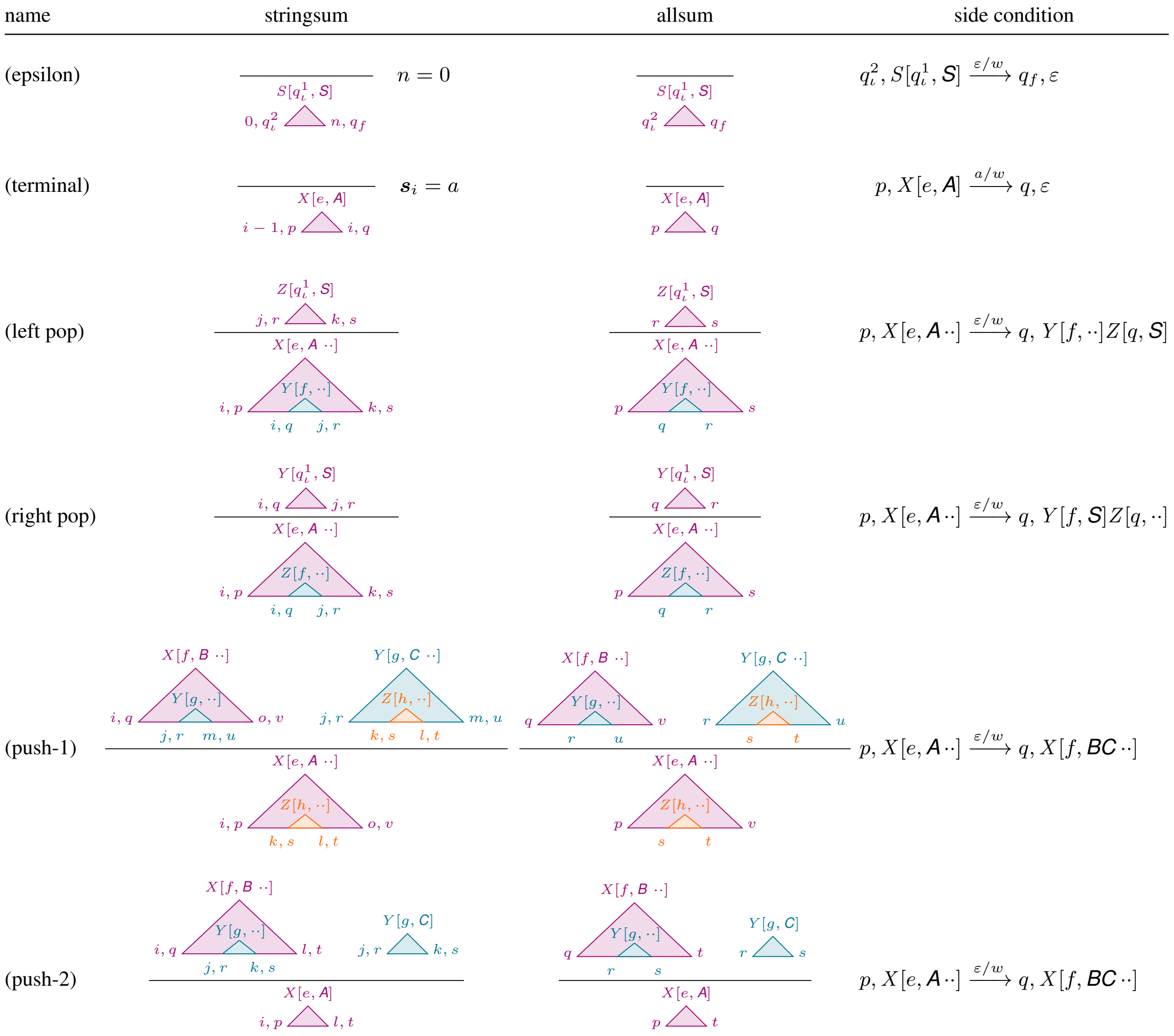
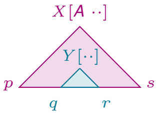

name
stringsum
allsum
side condition
(epsilon)
$n = 0$
$q_\iota^2, S[q_\iota^1, S] \xrightarrow{\varepsilon/w} q_f, \varepsilon$
(terminal)
$s_i = a$
$p, X[e, A] \xrightarrow{a/w} q, \varepsilon$
(left pop)
$p, X[e, A\,\cdot\cdot] \xrightarrow{\varepsilon/w} q, Y[f, \cdot\cdot]Z[q, S]$
(right pop)
$p, X[e, A\,\cdot\cdot] \xrightarrow{\varepsilon/w} q, Y[f, S]Z[q, \cdot\cdot]$
(push-1)
$p, X[e, A\,\cdot\cdot] \xrightarrow{\varepsilon/w} q, X[f, BC\,\cdot\cdot]$
(push-2)
$p, X[e, A\,\cdot\cdot] \xrightarrow{\varepsilon/w} q, X[f, BC\,\cdot\cdot]$

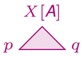
Figure 9: Deductive systems for computing stringsums and allsums of PDA ▷ PDA. States q_ι^1 and q_ι^2 are the initial states of $\mathbf{P}_1$ and $\mathbf{P}_2$, respectively. State q_f is the final state of $\mathbf{P}_2$.

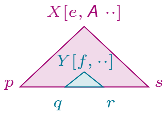
Definition 34. *Let* $\mathbf{G}$ *be a controller CFG with nonterminal alphabet* $\mathcal{N}$*, and let* $\mathbf{P}$ *be a WLD-PDA with states* Q *and stack alphabet* Γ*. A* ***pop computation*** *of* $\mathbf{G} \triangleright \mathbf{P}$ *of type*  *, where* $X, Y \in \Gamma$*,* $p, q, r, s \in Q$*, and* $A \in \mathcal{N}$*, is a pair of runs* $(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$*, where*

$$\boldsymbol{\pi}_1 \in ((p, X[A]) \overset{*}{\Rightarrow} (q, Y[\varepsilon]\boldsymbol{\Upsilon}_2)) \text{ and scans } \boldsymbol{s}_1$$

$$\boldsymbol{\pi}_2 \in ((r, \boldsymbol{\Upsilon}_2) \overset{*}{\Rightarrow} (s, \varepsilon)) \text{ and scans } \boldsymbol{s}_2$$

and $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \Sigma^*$*,* $\boldsymbol{\Upsilon}_2 \in \Gamma[\mathcal{N}^*]^*$*. The weight of this pop computation is* $\mathbf{w}(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = \mathbf{w}(\boldsymbol{\pi}_1) \otimes \mathbf{w}(\boldsymbol{\pi}_2)$*.*

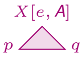
A ***pop computation*** *of type*  *is a run* $(p, X[A]) \overset{*}{\Rightarrow} (q, \varepsilon)$ *scanning some* $\boldsymbol{s} \in \Sigma^*$*.*

Definition 35. *Let* $\mathbf{P}_1$ *be a controller WPDA and* $\mathbf{P}_2$ *a WLD-PDA. Additionally, let* Q_1 *and* Γ_1*, and* Q_2 *and* Γ_2 *be the states and stack alphabets of* $\mathbf{P}_1$ *and* $\mathbf{P}_2$*, respectively. A* ***pop computation*** *of* $\mathbf{P}_1 \triangleright \mathbf{P}_2$ *of type*  *, where* $p, q, r, s \in Q_2$*,* $e, f \in Q_1$*,* $X, Y \in \Gamma_2$ *and* $A \in \Gamma_1$*, is is a pair of runs* $(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2)$*, where*

$$\boldsymbol{\pi}_1 \in ((p, X[e, A]) \overset{*}{\Rightarrow} (q, Y[f, \varepsilon]\boldsymbol{\Upsilon}_2)) \text{ and scans } \boldsymbol{s}_1$$

$$\boldsymbol{\pi}_2 \in ((r, \boldsymbol{\Upsilon}_2) \overset{*}{\Rightarrow} (s, \varepsilon)) \text{ and scans } \boldsymbol{s}_2$$

and $\boldsymbol{s}_1, \boldsymbol{s}_2 \in \Sigma^*$*,* $\boldsymbol{\Upsilon}_2 \in \Gamma_2[\Gamma_1^*]^*$*. The weight of this pop computation is* $\mathbf{w}(\boldsymbol{\pi}_1, \boldsymbol{\pi}_2) = \mathbf{w}(\boldsymbol{\pi}_1) \otimes \mathbf{w}(\boldsymbol{\pi}_2)$*.*

A ***pop computation*** *of type*  *is a run* $(p, X[e, A]) \overset{*}{\Rightarrow} (q, \varepsilon)$ *scanning some* $\boldsymbol{s} \in \Sigma^*$*.*

Figures 7 to 9, in the columns labeled "allsum", show deductive systems for computing allsums of PDA ▷ CFG, CFG ▷ PDA, and PDA ▷ PDA. As in §5, the value of the goal item can be computed by fixed-point iteration or the semiring generalization of Newton's method.

E Complexity Analysis of Conversions

Our stringsum and allsum algorithms, although designed for the two-level formalisms, can also be used for TAG, LIG, PAA, and EPDA. However, we must apply a series of conversions in order to do this. For instance, we must apply the following transformations to a WLIG with sets of nonterminals $\mathcal{N}$ and stack symbols Γ before feeding it into the stringsum algorithm:

1. The controller WPDA and the controllee WCFG have to be extracted from the WLIG, resulting in a WCFG with $\mathcal{O}(|\mathcal{N}|)$ nonterminals and a WPDA with $\mathcal{O}(|\Gamma|)$ stack symbols.

2. The controllee WCFG must be converted into Chomsky normal form. If the productions have at most k symbols on the right-hand side, then the output WCFG has $\mathcal{O}(k \times |\mathcal{R}|)$ nonterminals. Similarly, the controller WPDA must be converted into the normal form. If the WPDA has transitions that push at most k symbols, the resulting WPDA has $\mathcal{O}(k \times |\delta|)$ states.

3. The rules of the controller and of the controllee need to be merged back into a set of WLIG rules. This construction outputs a WLIG with $\mathcal{O}(|\mathcal{N}| \times |Q|)$ nonterminals and $\mathcal{O}(|\Gamma|)$ stack symbols, where $\mathcal{N}$ is the set of nonterminals of the input controllee, and Q and Γ are the sets of states and stack symbols of the input controller.