Convergence and Diversity in the Control Hierarchy

Alexandra Butoi[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

Abstract

Weir has defined a hierarchy of language classes whose second member ($\mathcal{L}_2$) is generated by tree-adjoining grammars (TAG), linear indexed grammars (LIG), combinatory categorial grammars, and head grammars. The hierarchy is obtained using the mechanism of *control*, and $\mathcal{L}_2$ is obtained using a context-free grammar (CFG) whose derivations are controlled by another CFG. We adapt Weir's definition of a controllable CFG to give a definition of controllable pushdown automata (PDAs). This yields three new characterizations of $\mathcal{L}_2$ as the class of languages generated by PDAs controlling PDAs, PDAs controlling CFGs, and CFGs controlling PDAs. We show that these four formalisms are not only weakly equivalent but equivalent in a stricter sense that we call *d-weak equivalence*. Furthermore, using an even stricter notion of equivalence called *d-strong equivalence*, we make precise the intuition that a CFG controlling a CFG is a TAG, a PDA controlling a PDA is an embedded PDA, and a PDA controlling a CFG is a LIG. The fourth member of this family, a CFG controlling a PDA, does not correspond to any formalism we know of, so we invent one and call it a Pushdown Adjoining Automaton.

1 Introduction

Weir (1992) defined a hierarchy of formal languages whose first level ($\mathcal{L}_1$) is the class of context-free languages, and whose second level ($\mathcal{L}_2$) is the class generated by several existing formalisms: tree-adjoining grammars (Joshi et al., 1975), linear indexed grammars (Gazdar, 1988), combinatory categorial grammars (Steedman, 1987) and head grammars (Pollard, 1984), which were proven weakly equivalent in a classic paper (Joshi et al., 1991; Vijay-Shanker and Weir, 1994), and embedded pushdown automaton (EPDA) as well (Vijay-Shanker, 1987).

Weir's hierarchy is obtained using the mechanism of *control*, and $\mathcal{L}_2$ is obtained by using a context-free grammar (CFG) to control another modified CFG, called a *labeled distinguished CFG* or LD-CFG. Here, we define a controllable pushdown automaton (PDA), called a *labeled distinguished PDA* (LD-PDA), and show that PDAs can also be used as controllers. By combining a controller CFG or PDA with a controlled LD-CFG or LD-PDA, we obtain a total of four formalisms that characterize $\mathcal{L}_2$.

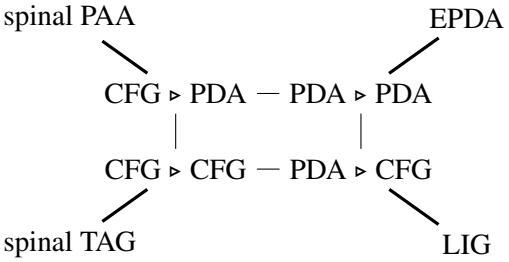


Figure 1: Overview of results. "$X \triangleright Y$" means "X controlling Y." Thin lines indicate d-weak equivalence, while thick lines indicate d-strong equivalence.

We show that these four formalisms are not only weakly equivalent but equivalent in a stricter sense that we call d-weak equivalence. Furthermore, using an even stricter notion of equivalence called d-strong equivalence, we make precise the intuition that a CFG controlling a CFG is essentially TAG, a PDA controlling a PDA is an EPDA, and a PDA controlling a CFG is a LIG. The fourth member of this family, CFGs controlling PDAs, does not correspond to any existing automaton we know of, so we invent one and call it a Pushdown Adjoining Automaton (PAA).

The main contributions of this paper are:

- Adapting Weir's LD-CFG to pushdown automata (LD-PDA).
- Two new definitions of equivalence between formalisms, d-weak equivalence and d-strong equivalence.
- Four d-weakly equivalent formalisms (one old, three new) obtained by controlling an LD-

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 1: Long Papers, pages 7597–7616

CFG/PDA using a controller CFG/PDA, three of which are d-strongly equivalent to TAG, LIG and EPDA, respectively.
- A new formalism, PAA, d-strongly equivalent to the fourth formalism.

2 Preliminaries

Let $[n]$ denote the set of integers $\{1, \ldots, n\}$, and let $[i{:}j]$ denote the set of integers $\{i, \ldots, j\}$.

For any sets $\mathcal{X}$ and $\mathcal{Y}$, we write $\breve{\mathcal{X}} = \{\breve{X} \mid X \in \mathcal{X}\}$ and $\mathcal{X}[\mathcal{Y}] = \{X[Y] \mid X \in \mathcal{X}, Y \in \mathcal{Y}\}$.

2.1 Context-free grammars

We assume familiarity with CFGs; see App. A.1 for definitions. The following normal form will be convenient later.

Definition 1. *A CFG is in **normal form** if its productions have one of the forms $X \to Y_1 \cdots Y_k$ or $X \to a$.*

A CFG derivation is sometimes thought of as a sequence of rewriting steps, and sometimes as a tree. The distinction is important in this paper, and we always refer to the former as *derivations* and the latter as *derivation trees*. See App. A.1, Def. 29 for a definition and Fig. 2a for an example.

2.2 Pushdown automata

Definition 2. *A **pushdown automaton** (PDA) is a tuple $\boldsymbol{P} = (Q, \Sigma, \Gamma, \delta, (q_\iota, S), (q_f, \varepsilon))$ where Q, Σ and Γ are finite sets of* states, input symbols *and* stack symbols, *δ is a finite set of* transitions, *and (q_ι, S) and (q_f, ε) are called the* initial *and* final configurations, *where $q_\iota, q_f \in Q$ and $S \in \Gamma$. The transitions are of the form $p, X \xrightarrow{a} q, \boldsymbol{\gamma}$, where $p, q \in Q$, $\boldsymbol{\gamma} \in \Gamma^*$, $X \in \Gamma$ and $a \in \Sigma \cup \{\varepsilon\}$. We say that a transition* scans *a and is* scanning *iff $|a| > 0$.*

This definition is similar to that of Hopcroft et al. (2006) but has only a single final configuration. Stacks are represented as strings over Γ, from top to bottom. Thus, in the stack $\boldsymbol{\gamma} = X_1 \cdots X_n$, the X_1 is at the top of the stack, while X_n is at the bottom.

See App. A.2 for further definitions. We again define a normal form that will be convenient later.

Definition 3. *A PDA is in **normal form** if its transitions have one of the forms $q, X \xrightarrow{\varepsilon} \boldsymbol{\gamma}$ or $q, X \xrightarrow{a} \varepsilon$.*

Unlike with CFGs, there is essentially no difference between treating PDA derivations as sequences of transitions or as (unary-branching) trees. See App. A.2, Def. 34 for a definition and Fig. 2b for an example.

3 Equivalence of Formalisms

We think of grammars and automata generically as "formal systems" that have derivation trees yielding strings. By equipping formal systems with derivation trees, we can define a notion of equivalence that is stricter than weak equivalence, and another notion that is more precise than strong equivalence.

Definition 4. *A **formal system** $\boldsymbol{F}$ over alphabet Σ is a set D of **derivation trees** together with a **yield function** $Y \colon D \to \Sigma^*$ that defines a language $\mathcal{L}(\boldsymbol{F}) = \{Y(d) \mid d \in D\}$.*

Definition 5. *Two formal systems $\boldsymbol{F}$ and $\boldsymbol{F}'$ are **d-weakly equivalent** if there is a bijection ϕ between the derivation trees of $\boldsymbol{F}$ and $\boldsymbol{F}'$ such that for any derivation tree d of $\boldsymbol{F}$ that yields the string $\boldsymbol{s}$, the derivation tree $\phi(d)$ of $\boldsymbol{F}'$ also yields $\boldsymbol{s}$.*

Example 1. *Consider two CFGs $\boldsymbol{G}_{fin}$ and $\boldsymbol{G}_{inf}$:*

$$\boldsymbol{G}_{fin} = \{\mathrm{S} \to \mathrm{a}\} \qquad \boldsymbol{G}_{inf} = \{\mathrm{S} \to \mathrm{S}, \mathrm{S} \to \mathrm{a}\}.$$

These grammars are weakly equivalent, but not d-weakly equivalent, because $\boldsymbol{G}_{fin}$ has just one derivation of a*, but $\boldsymbol{G}_{inf}$ has infinitely many, so there is no bijection between their derivation sets.*

Although there is a standard method to remove unary productions $\mathrm{S} \to \mathrm{S}$, on weighted grammars this method requires the semiring of weights to be complete; it will not work with general semirings. D-weak equivalence captures this distinction.

Example 2. *Consider a CFG $\boldsymbol{G}_{\mathrm{aa}} = \{\mathrm{S} \to \mathrm{AA}; \mathrm{A} \to \mathrm{a}\}$ and PDA $\boldsymbol{P}_{\mathrm{aa}} = \{p, \mathrm{S} \xrightarrow{\varepsilon} p, \mathrm{A}A; p, \mathrm{A} \xrightarrow{\mathrm{a}} p, \varepsilon\}$. Both $\boldsymbol{G}_{\mathrm{aa}}$ and $\boldsymbol{P}_{\mathrm{aa}}$ derive just one string,* aa*, each with a single derivation. Thus, $\boldsymbol{G}_{\mathrm{aa}}$ and $\boldsymbol{P}_{\mathrm{aa}}$ are d-weakly equivalent.*

Strong equivalence is traditionally understood to mean that two formal systems generate the same sets of structural descriptions, but this notion cannot be made precise without defining what structural descriptions are. For our purposes, it will suffice to use derivation trees, thought of as unlabeled, unordered trees.

Definition 6. *Two formal systems $\boldsymbol{F}$ and $\boldsymbol{F}'$ are **d-strongly equivalent** if there is a bijection ϕ between the derivation trees of $\boldsymbol{F}$ and $\boldsymbol{F}'$ such that for any derivation tree d of $\boldsymbol{F}$, d and $\phi(d)$ are isomorphic as unlabeled, unordered trees, and d and $\phi(d)$ yield the same string.*

Example 3. *Fig. 2 shows the derivation trees of the string* aa *in the CFG $\boldsymbol{G}$ and the PDA $\boldsymbol{P}$ defined*

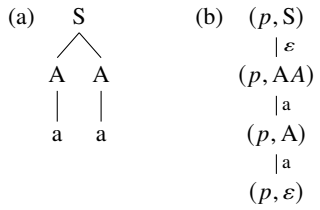


Figure 2: Example derivation trees for Example 2. (a) For CFG **G**, the nodes are nonterminals/terminals. (b) For PDA **P**, the nodes are configurations.

*in Example 2. **G** and **P** are* not *d-strongly equivalent, because there is no isomorphism between their derivation trees: the derivation tree in **G** is branching while the derivation tree in **P** is not.*

In general, CFG and PDA are d-weakly equivalent (App. D, Prop. 7), but not d-strongly equivalent.

4 Control

After a number of formalisms were proven to be equivalent characterizations of a language class $\mathcal{L}_2$ that is slightly more powerful than context-free languages (CFLs), Weir (1992) introduced the control hierarchy to characterize an infinite sequence of language classes, starting with CFLs and $\mathcal{L}_2$. The essential idea is that $\mathcal{L}_2$ languages can be recognized by a CFG in which the productions do not apply freely, but are *controlled* by another CFG.

4.1 LD-CFGs

To make a CFG controllable, Weir augments it slightly, as follows.

Definition 7. *A **labeled distinguished context-free grammar** (LD-CFG) is a tuple $\boldsymbol{G} = (\mathcal{N}, \Sigma, L, \mathcal{R}, S)$, where $\mathcal{N}$, Σ and L are finite sets of* nonterminal symbols, terminal symbols *and* labels, *$\mathcal{R}$ is a finite set of* productions, *and $S \in \mathcal{N}$ is the* start symbol. *The productions have one of the forms $\ell : A \to \alpha\check{B}\beta$ or $\ell : A \to \alpha$, where $\ell \in L$, $A \in \mathcal{N}$, $\check{B} \in \check{\mathcal{N}}$, and $\alpha, \beta \in (\mathcal{N} \cup \Sigma)^*$.*

The labels enable a *controller*, which is a formal system over L, to decide what productions the LD-CFG (the *controllee*) can use. Because the controller generates/accepts strings over L, whereas an LD-CFG derivation tree has a branching structure, the distinguishing marks ($\check{B}$) determine the paths of the derivation tree that are controlled.

Definition 8. *A **sentential form** of an LD-CFG is a pair $(\boldsymbol{\alpha}, W)$ where $\boldsymbol{\alpha} \in ((\mathcal{N} \cup \Sigma)[L^*])^*$ is a sequence of terminal and nonterminal symbols, each augmented with a string of labels, and $W \subseteq L^*$ is a set of **control words**.*

Definition 9. *If there is a production $\ell : A \to \boldsymbol{\beta}_1\check{B}\boldsymbol{\beta}_2 \in \mathcal{R}$, then we write $(\boldsymbol{\alpha}_1 A[w]\, \boldsymbol{\alpha}_2, W) \overset{\ell}{\Rightarrow} (\boldsymbol{\alpha}_1\, \boldsymbol{\beta}_1[\varepsilon]\, B[w\ell]\, \boldsymbol{\beta_2}[\varepsilon]\, \boldsymbol{\alpha}_2, W)$.*

If there is a production $\ell\colon A \to \boldsymbol{\beta}$, where $\boldsymbol{\beta} \in \mathcal{N}^*$, then we write $(\boldsymbol{\alpha}_1 A[w]\, \boldsymbol{\alpha}_2, W) \overset{\ell}{\Rightarrow} (\boldsymbol{\alpha}_1\, \boldsymbol{\beta}[\varepsilon]\, \boldsymbol{\alpha}_2, W \cup \{w\ell\})$.

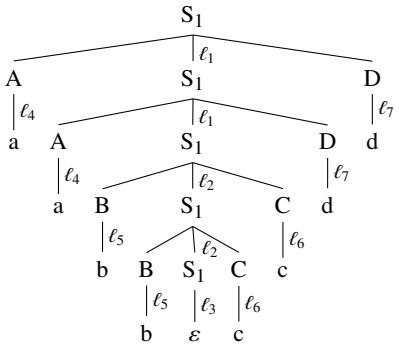
Definition 10. *Let $W \subseteq L^*$. If*

$$(S[\varepsilon], \emptyset) \overset{\ell_1}{\Longrightarrow} (\boldsymbol{\alpha}_1, W_1) \overset{\ell_2}{\Longrightarrow} \cdots \Rightarrow \ell_n(\boldsymbol{\alpha}_n, W_n)$$

*where $\boldsymbol{\alpha}_n \in \Sigma^*$ and $W_n \subseteq W$, we say that **G derives** $\boldsymbol{\alpha}_n$ under W.*

Example 4. *Below is an LD-CFG $\boldsymbol{G}_1$ and the derivation tree for the string* aabbccdd.

ℓ_1: $S_1 \to A\check{S}_1D$
ℓ_2: $S_1 \to B\check{S}_1C$
ℓ_3: $S_1 \to \varepsilon$
ℓ_4: $A \to a$
ℓ_5: $B \to b$
ℓ_6: $C \to c$
ℓ_7: $D \to d$



The control words used in this derivation are $\{\ell_1\ell_1\ell_2\ell_2\ell_3, \ell_4, \ell_5, \ell_6, \ell_7\}$.

Definition 11. *Let $W \subseteq L^*$. The **language** of **G** under W is the set of strings $\mathcal{L}(W, \boldsymbol{G}) = \{\boldsymbol{s} \mid$ **G** derives **s** under $W\}$.*

Definition 12. *Let $\boldsymbol{F}_2$ be a formal system defining strings over L and $\boldsymbol{G}_1$ an LD-CFG. Then $\boldsymbol{F}_2$ controlling $\boldsymbol{G}_1$ form a single formal system, which we call $\boldsymbol{F}_2 \triangleright \boldsymbol{G}_1$ and which defines the language $\mathcal{L}(\boldsymbol{F}_2 \triangleright \boldsymbol{G}_1) = \mathcal{L}(\mathcal{L}(\boldsymbol{F}_2), \boldsymbol{G}_1)$.*

Example 5. *Below is an example CFG $\boldsymbol{G}_2$ that can be used as a controller for $\boldsymbol{G}_1$ above (Example 4).*

$$\begin{aligned} &\mathsf{S}_2 \to \mathsf{T}\mathsf{L}_3 \\ &\mathsf{T} \to \mathsf{L}_1\mathsf{T}\mathsf{L}_2 \qquad\qquad \mathsf{T} \to \varepsilon \\ &\mathsf{L}_i \to \ell_i \quad (i \in [1{:}3]) \\ &\mathsf{S}_2 \to \ell_i \quad (i \in [4{:}7]) \end{aligned}$$

It generates the language $\{\ell_1^n\ell_2^n\ell_3 \mid n \geq 0\} \cup \{\ell_i \mid i \in [4{:}7]\}$, which makes the LD-CFG generate the language $\{\mathrm{a}^n\mathrm{b}^n\mathrm{c}^n\mathrm{d}^n \mid n \geq 0\}$.

The controller could also be a PDA, like $\boldsymbol{P}_2$:

$$q, \mathsf{S}_2 \xrightarrow{\varepsilon} q, \mathsf{TL}_3$$

$$q, \mathsf{T} \xrightarrow{\varepsilon} q, \mathsf{L}_1\mathsf{TL}_2 \qquad q, \mathsf{T} \xrightarrow{\varepsilon} q, \varepsilon$$

$$q, \mathsf{L}_i \xrightarrow{\ell_i} q, \varepsilon \ (i \in [1{:}3])$$

$$q, \mathsf{S}_2 \xrightarrow{\ell_i} q, \varepsilon \ (i \in [4{:}7])$$

4.2 LD-PDAs

Weir's definition of LD-CFG can be adapted to pushdown automata.

Definition 13. *A **labeled distinguished PDA** (LD-PDA) is a tuple $\boldsymbol{P} = (Q, \Sigma, \Gamma, L, \delta, (q_\iota, S), (q_f, \boldsymbol{\gamma}_f))$, where Q, Σ and Γ are finite sets of* states, input symbols *and* stack symbols, *δ is a finite set of* transitions, *and (q_ι, S) and $(q_f, \boldsymbol{\gamma}_f) \in Q \times (\Gamma[L^*])^*$ are the* initial *and* final configurations. *The transitions are of the form $\ell\colon q, A \xrightarrow{a} r, \boldsymbol{\alpha}$, or $\ell\colon q, A \xrightarrow{a} r, \boldsymbol{\alpha}\breve{B}\boldsymbol{\beta}$, where $\ell \in L$, $q, r \in Q$, $a \in \Sigma$, $A \in \Gamma$, $\breve{B} \in \breve{\Gamma}$, and $\boldsymbol{\alpha}, \boldsymbol{\beta} \in \Gamma^*$.*

Example 6. *Below is an example LD-PDA, $\boldsymbol{P}_1$, where q is both initial and final:*

$$\ell_1\colon q, \mathrm{S}_1 \xrightarrow{\varepsilon} q, \mathrm{A}\breve{\mathrm{S}}_1\mathrm{D} \quad \ell_2\colon q, \mathrm{S}_1 \xrightarrow{\varepsilon} q, \mathrm{B}\breve{\mathrm{S}}_1\mathrm{C}$$

$$\ell_3\colon q, \mathrm{S}_1 \xrightarrow{\varepsilon} q, \varepsilon$$

$$\ell_4\colon q, \mathrm{A} \xrightarrow{\mathrm{a}} q, \varepsilon \qquad \ell_5\colon q, \mathrm{B} \xrightarrow{\mathrm{b}} q, \varepsilon$$

$$\ell_6\colon q, \mathrm{C} \xrightarrow{\mathrm{c}} q, \varepsilon \qquad \ell_7\colon q, \mathrm{D} \xrightarrow{\mathrm{d}} q, \varepsilon$$

Definition 14. *A **configuration** of an LD-PDA $\boldsymbol{P}$ is a tuple $(q, \boldsymbol{\gamma}, W)$, where q is the current state, $\boldsymbol{\gamma} \in (\Gamma[L^*])^*$ is a sequence of stack symbols, each augmented with a string of labels, and $W \subseteq L^*$ is a set of control words.*

Definition 15. *If there is a transition $\ell\colon q, A \xrightarrow{a} r, \boldsymbol{\beta}_1\breve{B}\boldsymbol{\beta}_2 \in \delta$, we write $(q, A[w]\,\boldsymbol{\gamma}, W) \overset{\ell}{\Longrightarrow} (r, \boldsymbol{\beta}[\varepsilon]\,B[w\ell]\,\boldsymbol{\beta}[\varepsilon]\,\boldsymbol{\gamma}', W)$.*

If there is a transition $\ell\colon q, A \overset{a}{\Rightarrow} r, \boldsymbol{\beta}$, we write $(q, A[w]\,\boldsymbol{\gamma}, W \cup \{w\ell\}) \overset{\ell}{\Rightarrow} (r, \boldsymbol{\beta}[\varepsilon]\,\boldsymbol{\gamma}, W \cup \{w\ell\})$.

Definition 16. *Let $W \subseteq L^*$. If $(q_\iota, S, \emptyset) \overset{\ell_1}{\Longrightarrow} (q_1, \boldsymbol{\gamma}_1, W_1) \overset{\ell_2}{\Longrightarrow} \cdots \overset{\ell_{n-1}}{\Longrightarrow} (q_{n-1}, \boldsymbol{\gamma}_{n-1}, W_{n-1}) \overset{\ell_n}{\Longrightarrow} (q_f, \varepsilon, W_n)$, where $W_n \subseteq W$, and for $i = 1, \ldots, n$, transition ℓ_i scans a_i, then we say that $\boldsymbol{P}$ **accepts** string $a_1 a_2 \cdots a_{n-1} a_n$ under W.*

Definition 17. *Let $W \subseteq L^*$. The **language** accepted by $\boldsymbol{P}$ under W is the set of strings $\mathcal{L}(W, \boldsymbol{P}) = \{\boldsymbol{s} \mid \boldsymbol{P}$ accepts $\boldsymbol{s}$ under $W\}$.*

Definition 18. *Let $\boldsymbol{F}_2$ be a formal system defining strings over L and $\boldsymbol{P}_1$ an LD-PDA. Then $\boldsymbol{F}_2$ controlling $\boldsymbol{P}_1$ form a single formal system, which we call $\boldsymbol{F}_2 \triangleright \boldsymbol{P}_1$ and which defines the language $\mathcal{L}(\boldsymbol{F}_2 \triangleright \boldsymbol{P}_1) = \mathcal{L}(\mathcal{L}(\boldsymbol{F}_2), \boldsymbol{P}_1)$.*

4.3 Four two-level formalisms

Considering both CFGs and PDAs as both controllers and controllees yields four two-level formalisms, one of which is Weir's original two-level grammar, and the other three of which are new.

Proposition 1. *CFG ▷ CFG, CFG ▷ PDA, PDA ▷ PDA and PDA ▷ CFG are d-weakly equivalent.*

Proof. See App. D. □

From now on, we will use different fonts for the symbols in the controllee and the controller. In a controllee LD-CFG/LD-PDA, we use $X, Y, \ldots$ and $a, b, \ldots$ for the nonterminal/stack symbols and terminal/input symbols. In a controller CFG/PDA, we use $\mathsf{A}, \mathsf{B}, \ldots$ for the nonterminal/stack symbols and $\mathsf{a}, \mathsf{b}, \ldots$ or $\ell, \ell_1, \ldots$ for the terminal/input symbols.

5 Derivation Trees of Two-Level Formalisms

In order to show that our four two-level formalisms are d-strongly equivalent to other $\mathcal{L}_2$ formalisms, we need to establish what derivation trees look like in these formalisms. Weir actually gives two definitions of derivation in a two-level grammar. The first is the one we have reproduced above, where a partial control word is written inside the square brackets. The second runs a leftmost derivation of the controller CFG inside the square brackets. In this section, we follow this second approach, except that we use derivation trees, not leftmost derivations. We give a separate definition for each two-level formalism. Ideally, these definitions would have followed "for free" from the definitions of derivation trees of (LD-)CFGs and (LD-)PDAs, but we save this level of generality for future work.

Fig. 3 shows the derivation of string aabbccdd under the four two-level formalisms considered in this paper. The rest of this section discusses each formalism in turn. We assume all controllers and controllees are in normal form; see App. C for the general case.

PDA ▷ PDA For PDA, derivations and derivation trees are the same thing, so derivation trees of PDA

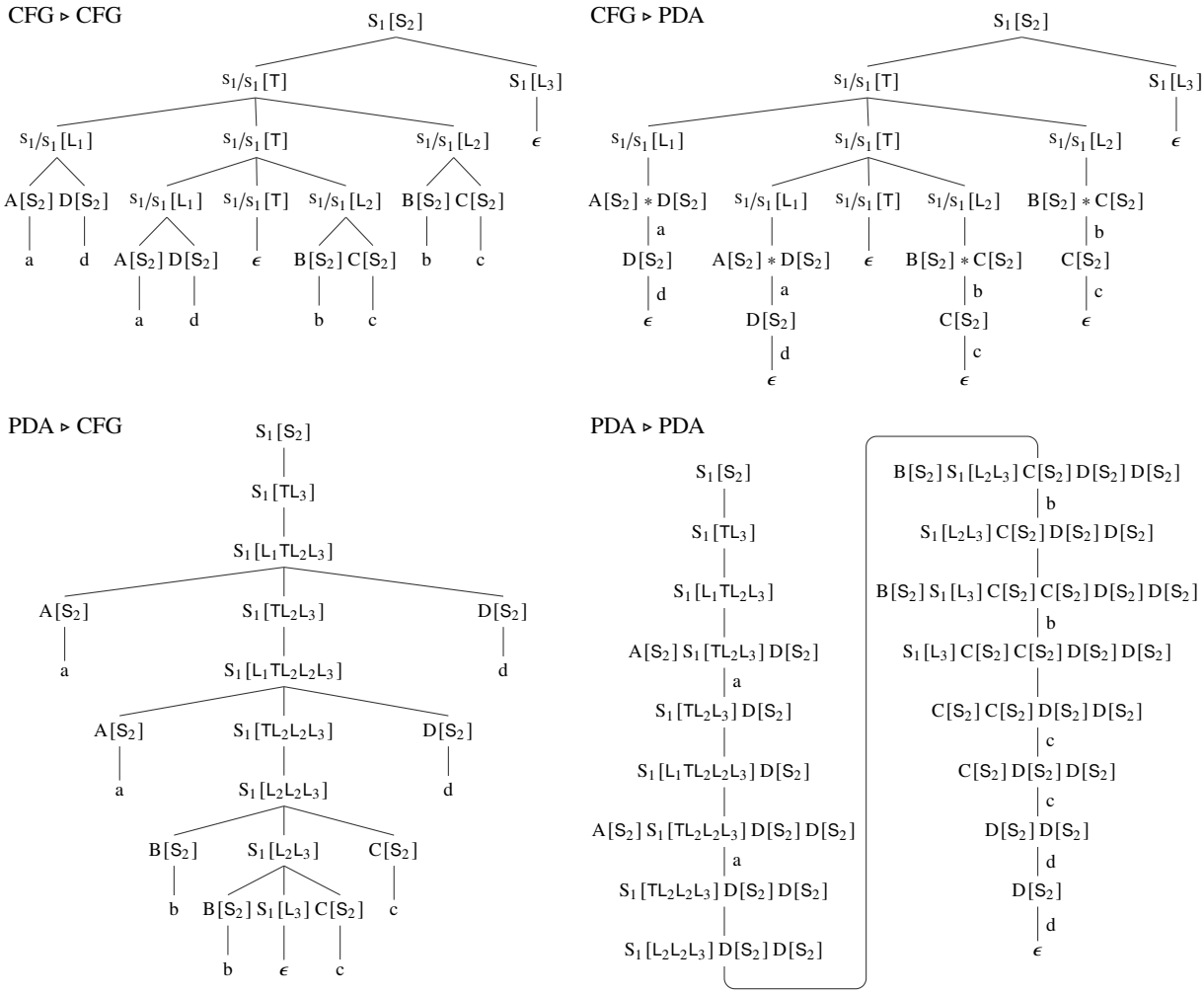


Figure 3: Derivation trees of aabbccdd under the four two-level formalisms considered in this paper. For pushdown automata, states are omitted.

▷ PDA follow Weir's definition straightforwardly. There are three cases to consider.

(a) If a derivation node is labeled $q, X[s, A]\,\Upsilon$, and $s, A \xrightarrow{\ell} q_f, \varepsilon$ is a controller transition, and ℓ is a controllee transition $q, X \xrightarrow{a} r, \varepsilon$, then the node can have child

$$\begin{array}{c} q, X[s, A]\,\Upsilon \\ | a \\ r, \Upsilon \end{array}$$

(b) If a derivation node is labeled $q, X[s, A\boldsymbol{\beta}]\,\Upsilon$, and $s, A \xrightarrow{\ell} t, \varepsilon$ is a controller transition, and ℓ is a controllee transition $q, X \xrightarrow{\varepsilon} r, Y_1 \cdots Y_{d-1}\,\breve{Z}\,Y_{d+1} \cdots Y_k$, then

$$\begin{array}{c} q, X[s, A\boldsymbol{\beta}]\,\Upsilon \\ | \\ r, Y_1[q_\iota, S_2] \cdots Y_{d-1}[q_\iota, S_2]\,Z[t, \boldsymbol{\beta}]\,Y_{d+1}[q_\iota, S_2] \cdots Y_k[q_\iota, S_2]\,\Upsilon \end{array}$$

(c) If a derivation node is labeled $q, X[s, A\boldsymbol{\beta}]\,\Upsilon$, and $s, A \xrightarrow{\varepsilon} t, \boldsymbol{\gamma}$ is a controller transition, then

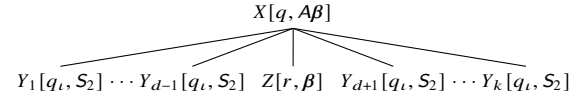
$$\begin{array}{c} q, X[s, A\boldsymbol{\beta}]\,\Upsilon \\ | \\ q, X[t, \boldsymbol{\gamma\beta}]\,\Upsilon \end{array}$$

PDA ▷ CFG This formalism is not much more difficult than CFG ▷ CFG.

(a) If a derivation node is labeled $X[q, A]$, and $q, A \xrightarrow{\ell} q_f, \varepsilon$ is a controller transition, and ℓ is a controllee production $X \to a$, then

$$\begin{array}{c} X[q, A] \\ | \\ a \end{array}$$

(b) If a derivation node is labeled $X[q, A\boldsymbol{\beta}]$, and $q, A \xrightarrow{\ell} r, \varepsilon$ is a controller transition, and ℓ is a controllee production $X \to Y_1 \cdots Y_{d-1}\,\breve{Z}\,Y_{d+1} \cdots Y_k$, then



(c) If a derivation node is labeled $X[q, A\boldsymbol{\beta}]$, and $q, A \xrightarrow{\varepsilon} r, \boldsymbol{\gamma}$ is a controller transition, then

$$\begin{array}{c} X[q, A\boldsymbol{\beta}] \\ | \\ X[r, \boldsymbol{\gamma}\boldsymbol{\beta}] \end{array}$$

CFG ▷ CFG When the controller is a CFG and therefore has branching derivation trees, the controllee's derivation becomes fragmented. To ensure that the fragments form one correct derivation, we need to add some extra information to the derivation nodes: each node is labeled ${}^{X}/_{Z}[A]$, where A is a controller nonterminal, and X and Z are controllee nonterminals or $\bot$. We write ${}^{X}/_{\bot}[A]$ as $X[A]$.

Moreover, the yield of a subderivation is in general a discontinuous string. We write $*$ to indicate a hole in a string (meant to resemble foot nodes in TAG), and we write $u(v)$ to plug v into the first hole in u; that is, if $u = u_1 * u_2$ where u_1 does not have a hole, then $u(v) = u_1 v u_2$.

(a) If a derivation node is labeled $X[A]$, and $A \to \ell$ is a controller production, and ℓ is a controllee production $X \to a$, then

$$\begin{array}{c} X[A] \\ | \\ a \end{array}$$

The yield is a.

(b) If a derivation node is labeled ${}^{X}/_{Z}[A]$, and $A \to \ell$ is a controller production, and ℓ is a controllee production $X \to Y_1 \cdots Y_{d_{i-1}} \check{Z}\, Y_{d+1} \cdots Y_k$, then

$$\begin{array}{c} {}^{X}/_{Z}[A] \\ \diagup \quad \diagup \quad \diagdown \quad \diagdown \\ Y_1[S_2] \;\cdots\; Y_{d-1}[S_2] \quad Y_{d+1}[S_2] \;\cdots\; Y_k[S_2] \end{array}$$

If each child node $Y_i[S_2]$ has yield w_i, then the parent has yield $w_1 \cdots w_{d-1} * w_{d+1} \cdots w_k$.

(c) If a derivation node is labeled ${}^{X}/_{Z}[A]$, and $A \to B_1 \cdots B_k$ is a controller production, then

$$\begin{array}{c} {}^{X}/_{Z}[A] \\ \diagup \quad \diagdown \\ {}^{Y_0}/_{Y_1}[B_1] \quad \cdots \quad {}^{Y_{k-1}}/_{Y_k}[B_k] \end{array}$$

where $Y_0 = X$, $Y_k = Z$, and $Y_1, \ldots, Y_{k-1}$ can be any controllee nonterminals. If the children have yields $w_1, w_2, \ldots, w_k$, then the parent has yield $w_1(w_2(\cdots(w_k)))$.

CFG ▷ PDA The derivation trees of CFG ▷ PDA are the most complicated. To cut down on complexity, we make use of the fact that every PDA that accepts by empty stack is equivalent to one with only one state (Hopcroft et al., 2006), and indeed the equivalence is d-strong (App. A, Prop. 6). Therefore, we assume that the controllee has just one state, q.

Derivation nodes are labeled $(q, \boldsymbol{\Upsilon})$, where q is a state and $\boldsymbol{\Upsilon}$ is a string of symbols of the form ${}^{X}/_{Z}[A]$ (as above, for CFG ▷ CFG) or $*$.

(a) If a derivation node is labeled $q, *^{?}X[A]\,\boldsymbol{\Upsilon}$, where $*^{?}$ denotes either the presence or absence of a $*$, and $A \to \ell$ is a controller production, and ℓ is a controllee transition $q, X \xrightarrow{a} q, \varepsilon$, then

$$\begin{array}{c} q, *^{?}X[A]\,\boldsymbol{\Upsilon} \\ |\, a \\ q, \boldsymbol{\Upsilon} \end{array}$$

If the yield of the child is w, then the yield of the parent is $*^{?}aw$.

(b) If a derivation node is labeled $q, {}^{X}/_{Z}[A]\,\boldsymbol{\Upsilon}$, and $A \to \ell$ is a controller production, and ℓ is a controllee transition $q, X \xrightarrow{\varepsilon} q, Y_1 \cdots Y_{d-1} \check{Z}\, Y_{d+1} \cdots Y_k$, then

$$\begin{array}{c} q, {}^{X}/_{Z}[A]\,\boldsymbol{\Upsilon} \\ | \\ q, Y_1[S_2] \cdots Y_{d-1}[S_2] * Y_{d+1}[S_2] \cdots Y_k[S_2]\,\boldsymbol{\Upsilon} \end{array}$$

The yield of the parent is just the yield of the child.

(c) If a derivation node is labeled $q, {}^{X}/_{Z}[A]\,\boldsymbol{\Upsilon}$, and $A \to B_1 \cdots B_k$ is a controller production, then

$$\begin{array}{c} q, {}^{X}/_{Z}[A]\,\boldsymbol{\Upsilon} \\ \diagup \quad \diagdown \\ q, {}^{Y_0}/_{Y_1}[B_1]\,\boldsymbol{\Upsilon} \quad \cdots \quad q, {}^{Y_{k-1}}/_{Y_k}[B_k] \end{array}$$

where $Y_0 = X$, $Y_k = Z$, and $Y_1, \ldots, Y_{k-1}$ can be any controllee nonterminals. If the children have yields $w_1, w_2, \ldots, w_k$, then the parent has yield $w_1(w_2(\cdots(w_k)))$.

The reader may be surprised that $\boldsymbol{\Upsilon}$ is placed on the leftmost child. It would be equally sensible to place it on any of the children, but the leftmost child makes the yield function simplest.

6 Correspondences with $\mathcal{L}_2$ Formalisms

We can now show how PDA ▷ PDA, PDA ▷ CFG, and CFG ▷ CFG are d-strongly equivalent to EPDA, LIG, and TAG, respectively, and to introduce PAA, which is d-strongly equivalent to CFG ▷ PDA.

6.1 Embedded pushdown automata

Definition 19. *An **embedded pushdown automaton** (EPDA) is a tuple*

$\boldsymbol{P} = (Q, \Sigma, \Gamma, \delta, (q_\iota, [S]), (q_f, \varepsilon))$, *where* Q, Σ, $\mathcal{N}$ *and* Γ *are finite sets of* states, input symbols *and* stack symbols, δ *is a finite set of* transitions, *and* $(q_\iota, [S])$ *and* (q_f, ε) *are called the* initial *and* final configurations, *where* $q_\iota, q_f \in Q$, *and* $S \in \Gamma$. *The transitions are of the form* $p, [A \cdots] \xrightarrow{a} q, \Upsilon[\gamma \cdots]\Upsilon'$ *or* $p, [A] \xrightarrow{a} q, \varepsilon$, *where* $p, q \in Q$, $A \in \Gamma$, $\gamma \in \Gamma^*$, $\Upsilon, \Upsilon' \in ([S])^*$, *and* $a \in \Sigma \cup \{\varepsilon\}$.

An EPDA maintains a stack of stacks. Each transition can pop and push from the top stack, as well as pop the top stack and/or push other stacks above and below it.

Proposition 2. *An EPDA is d-strongly equivalent to a PDA* ▷ *PDA.*

Proof. See App. E. □

6.2 Linear indexed grammars

Definition 20. *A* ***linear indexed grammar*** *(LIG) is a tuple* $\boldsymbol{G} = (\mathcal{N}, \Sigma, \Gamma, \mathcal{R}, S, S)$, *where* $\mathcal{N}$, Σ *and* Γ *are sets of* nonterminals, terminals *and* stack symbols, $\mathcal{R}$ *is a finite set of* productions, $S \in \mathcal{N}$ *is the* start symbol, *and* $S \in \Gamma$ *is the* initial stack. *The production rules are of the form* $X[A \cdots] \to \Upsilon_1 Y[\gamma \cdots]\Upsilon_2$ *or* $X[A] \to a$, *where* $X, Y \in \mathcal{N}$, $\gamma \in \Gamma^*$, $A \in \Gamma$, $\Upsilon_1, \Upsilon_2 \in (\mathcal{N}[S] \cup \Sigma)^*$, *and* $a \in \Sigma \cup \{\varepsilon\}$.

Each nonterminal symbol is augmented with a stack (in square brackets). A production with lhs X rewrites a nonterminal $X[A\gamma]$ into its rhs, while popping A and pushing zero or more stack symbols.[1] Exactly one of the rhs nonterminals inherits the stack, while the others get new stacks.

Proposition 3. *A LIG is d-strongly equivalent to a PDA* ▷ *CFG.*

Proof. See App. E. □

6.3 Tree-adjoining grammars

Definition 21. *A* ***tree-adjoining grammar*** *(TAG) is a tuple* $\boldsymbol{G} = (\mathcal{V}, C, \Sigma, S, \mathcal{R})$ *where* $\mathcal{V}$, C *and* Σ *are finite sets of* variable, constant,[2] *and* terminal symbols, $S \in \mathcal{V}$ *is the* start symbol *and* $\mathcal{R}$ *is a set of* productions. *The productions are of the form* $X \to \beta$, *where* $X \in \mathcal{V}$ *and* $\beta \in \mathcal{T}(\mathcal{V} \cup C \cup \Sigma \cup \{\varepsilon\})$. *In* β, *interior nodes must be in* $\mathcal{V} \cup C$. *There is one leaf in* $\mathcal{V} \cup C$ *that is designated the* ***foot node*** *and marked with* $*$. *The path from the root to the foot node is called the* ***spine***. *Other leaves must be in* $\mathcal{V}$ *(in which case they are called* ***substitution nodes*** *and marked with* $\downarrow$*) or* $\Sigma \cup \{\varepsilon\}$.

This definition is close to *non-strict TAG* (Lang, 1994; Rogers, 2003). Unlike the usual definition, its rules have left-hand sides and do not require the root and foot node to have the same label. We also do not allow substitution, as it can be simulated by adjunction. A TAG derivation starts with S and proceeds by repeatedly choosing a variable X and rewriting it using a production $X \to \beta$, such that the children of X become the children of β's foot node. See App. B.3 for additional definitions.

Definition 22. *A TAG production* $X \to \beta$, *where* β *has a foot node, is called* ***spinal***[3] *if every node of is either on the spine or the child of a node on the spine. A TAG production* $X \to \alpha$, *where* α *does not have a foot node, is called spinal if there is a leaf* η *such that every node is either on the path from the root to* η *or the child of such a node. A TAG is called spinal if all its productions are spinal.*

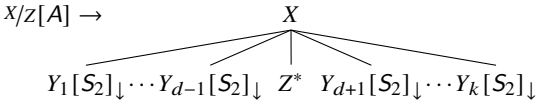
TAG derivation trees were defined by Vijay-Shanker (1987). We show an example TAG and derivation tree in Fig. 4.

Proposition 4. *A spinal TAG is d-strongly equivalent to a CFG* ▷ *CFG.*

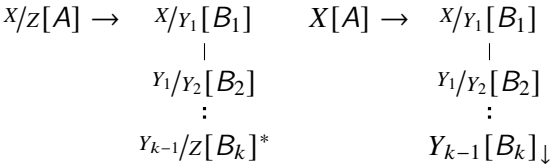
Proof. See App. E; we give a brief sketch here, using the same three cases as in §5. Case (a) corresponds to the TAG production

$$X[A] \to \begin{array}{c} X \\ | \\ a \end{array}$$

Case (b) corresponds to the TAG production



Case (c) corresponds to the TAG productions



[1] Most LIG definitions (Vijay-Shanker, 1987; Joshi et al., 1991; Kanazawa, 2014) allow only productions that pop/push at most one symbol from/to a nonterminal's stack.

[2] The terms *variable* and *constant* come from Lang (1994) and are equivalent to nonterminals with obligatory and null adjunction constraints, respectively.

[3] This term comes from Fujiyoshi and Kasai (2000), who use it for context-free tree grammars.

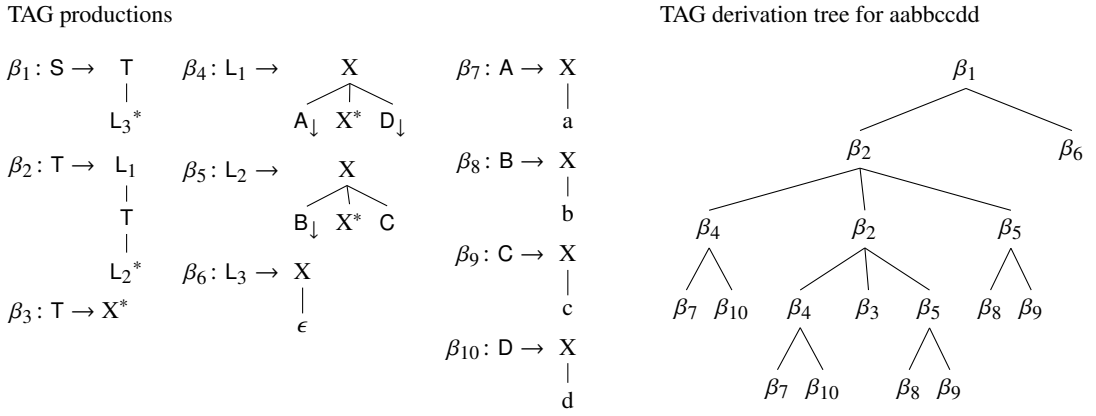


Figure 4: Example TAG and derivation tree for aabbccdd. To reduce clutter, symbols have been renamed. Symbol X is constant; other uppercase symbols are variable.

If $k = 0$ then the corresponding TAG productions are

$$X/X[A] \to X^* \qquad X[A] \to \begin{matrix} X \\ | \\ \varepsilon \end{matrix} \qquad \square$$

6.4 Pushdown adjoining automata

A pushdown adjoining automaton (PAA) maintains a stack of symbols. The stack symbols are either *variable* symbols, which can be rewritten, or *constant* symbols, which cannot. Each stack symbol is augmented with another stack, written below it:

$$\begin{matrix} ABC \\ \llcorner DE \end{matrix}$$

Here, the top-level stack is ABC, in which B is augmented with stack DE. The other symbols have empty stacks, which are not shown.

A PAA transition has a lhs which is a variable symbol and a rhs which is a stack (as described above). The rhs may have one symbol occurrence called the *foot*; if so, both the lhs and the foot are written as $\underset{\llcorner}{X}$.

At each step, the PAA operates on the topmost stack that has a variable X on top. If it has a transition $X \xrightarrow{a} \xi$, then it scans a, pops X and pushes ξ. If X had a stack attached, then the stack becomes attached to the foot of ξ. Finally, any constant symbol that has an empty stack is deleted.

Example 7. *The PAA for* $\{a^n b^n c^n d^n\}$ *is shown in Fig. 5, along with the run of this PAA on* aabbccdd.

The nested stacks of a PAA, described informally above, are treated formally as trees with a special constant symbol $\top$ at the root. However, we continue to draw them as shown above, without $\top$.

Definition 23. *A* ***pushdown adjoining automaton*** *(PAA) is a tuple* $\boldsymbol{P} = (\Sigma, \mathcal{V}, C, \delta, S)$ *where* Σ, $\mathcal{V}$, *and* C *are finite sets of* input symbols, variable stack symbols, *and* constant stack symbols, δ *is a set of* transitions *and* $S \in \mathcal{V}$ *is the* initial stack. *The transitions are of the form* $X \xrightarrow{a} \xi$ *where* $X \in \mathcal{V}$ *and* $\xi \in \mathcal{T}(\mathcal{V} \cup C)$. *The root of* ξ *is* $\top$, *and at most one leaf in* ξ *is designated the* foot *and written as* $\underset{\llcorner}{X}$.

Definition 24. *The path from the root to the foot of a PAA production's rhs is called its* ***spine****. We say that a PAA production is* ***spinal*** *if every rhs node is either on the spine or a child of a node on the spine. A PAA is* ***spinal*** *if all of its productions are spinal.*

Definition 25. *If* $\xi \in \mathcal{T}(\mathcal{V} \cup C)$ *without constant leaf nodes, the* ***top variable*** *of* ξ *is defined as follows: If* $\xi = X$, *then the top variable is* X. *If* $\xi = \begin{matrix} X \\ \llcorner \xi_1 \ldots \end{matrix}$ *where* $X \in \mathcal{V}$, *then the top variable is* X. *If* $\xi = \begin{matrix} X \\ \llcorner \xi_1 \ldots \end{matrix}$ *where* $X \in C$, *then the top variable of* ξ *is the top variable of* ξ_1.

Definition 26. *If* ξ *has top variable* X, *and there is a transition* $\tau = (X \xrightarrow{a} \rho)$, *then we write* $\xi \overset{\tau}{\Longrightarrow} \xi'$ *if* ξ' *can be obtained from* ξ *by replacing its top variable* X *with* ρ *(sans* $\top$*), making the children of* X *the children of the foot of* ρ, *and deleting any constant leaf nodes.*

In a PAA derivation tree (see Fig. 5 for an example), every node is a stack, written without its embedded stacks. If $X\boldsymbol{\gamma}$ is such a node and $(X \xrightarrow{a} \xi)$ is a transition, then the stacks of $\xi\boldsymbol{\gamma}$ are its children.

Proposition 5. *A spinal PAA is d-strongly equivalent to a CFG* ▷ *PDA.*

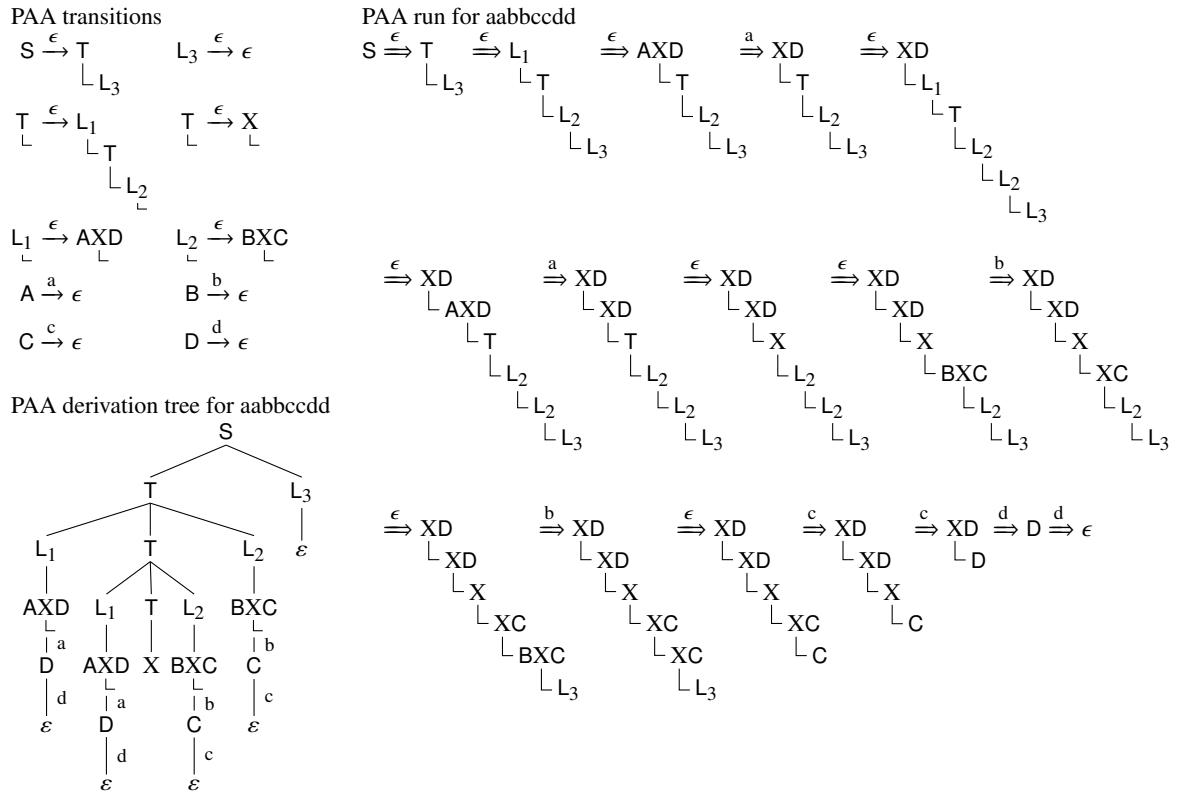


Figure 5: Example PAA with run and derivation tree for aabbccdd. To reduce clutter, symbols have been renamed. Symbol X is constant; other uppercase symbols are variable.

Proof. See App. E; we give a brief sketch here, using the same three cases as in §5.

Case (a) corresponds to the rule

$$X[A] \xrightarrow{a} \varepsilon$$

Case (b) corresponds to the rule

$$X/Z[A] \xrightarrow{\varepsilon} Y_1[S_2] \cdots Y_{d-1}[S_2]\, Z\, Y_{d+1}[S_2] \cdots Y_k[S_2]$$

where Z is constant.

Case (c) corresponds to the rules

$$X/Z[A] \xrightarrow{\varepsilon} X/Y_1[B_1]$$
$$\qquad\qquad\qquad\quad \llcorner\, Y_1/Y_2[B_2]$$
$$\qquad\qquad\qquad\qquad\quad \vdots$$
$$\qquad\qquad\qquad\qquad\quad \llcorner\, Y_{k-1}/Z[B_k]$$

$$X[A] \xrightarrow{\varepsilon} X/Y_1[B_1]$$
$$\qquad\qquad\qquad\quad \llcorner\, Y_1/Y_2[B_2]$$
$$\qquad\qquad\qquad\qquad\quad \vdots$$
$$\qquad\qquad\qquad\qquad\quad \llcorner\, Y_{k-1}[B_k]$$

If $k = 0$ then we must insert a constant symbol:

$$X/X[A] \xrightarrow{\varepsilon} X \qquad X[A] \xrightarrow{\varepsilon} X \qquad \square$$

7 Conclusion

We introduced new notions of equivalence between formalisms, d-weak equivalence and d-strong equivalence, that allow for finer-grained comparisons than weak equivalence. By extending Weir's idea of control to PDAs, we obtained three new formalisms recognizing $\mathcal{L}_2$, all d-weakly equivalent to Weir's original two-level grammar. But by nuancing the idea of control to account for the difference between CFG derivations and derivation trees, we showed that they are not d-strongly equivalent. Instead, three of them are d-strongly equivalent to existing $\mathcal{L}_2$ formalisms, namely TAG, LIG and EPDA, and the fourth is d-strongly equivalent to our new $\mathcal{L}_2$ formalism, PAA.

Limitations

We currently give separate definitions of the derivation trees of the four two-level formalisms. Ideally, one could give a general definition of derivation tree in a multi-level formalism, and then derive each particular case by plugging in the definition of derivation trees of (LD-)CFGs and (LD-)PDAs. We leave this generalization for future work.

The d-strong equivalence results only hold for spinal TAG and spinal PAA. However, full TAG and PAA are not d-strongly equivalent to spinal TAG and spinal PAA, and therefore they are not d-strongly equivalent to CFG ▷ CFG and CFG ▷ PDA, either.

Ethics Statement

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

Acknowledgements

We would like to thank the anonymous reviewers for their comments and suggestions.

References

Akio Fujiyoshi and Takumi Kasai. 2000. Spinal-formed context-free tree grammars. *Theory of Computing Systems*, 33(1):59–83.

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

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

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

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

Bernard Lang. 1994. Recognition can be harder than parsing. *Computational Intelligence*, 10(4):486–494.

Carl J. Pollard. 1984. *Generalized Phrase Structure Grammars, Head Grammars and Natural Language*. Ph.D. thesis, Stanford University.

James Rogers. 2003. wMSO theories as grammar formalisms. *Theoretical Computer Science*, 293(2):291–320.

Mark Steedman. 1987. Combinatory grammars and parasitic gaps. *Natural Language & Linguistic Theory*, 5(3):403–439.

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

K. Vijay-Shanker and David Weir. 1994. The equivalence of four extensions of context-free grammars. *Mathematical Systems Theory*, 27(6):511–546.

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

A Context-Free Formal Systems

A.1 Context-Free Grammars

Definition 27. *A **context-free grammar** (CFG) is a tuple $\boldsymbol{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$, where $\mathcal{N}$ and Σ are finite sets of* nonterminal *and* terminal symbols*, $\mathcal{R}$ is a finite set of* production rules*, and $S \in \mathcal{N}$ is a distinguished* start symbol*. The production rules are of the form $A \to \boldsymbol{\alpha}$, where $A \in \mathcal{N}$ and $\boldsymbol{\alpha} \in (\mathcal{N} \cup \Sigma)^*$.*

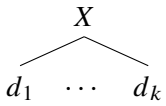
Definition 28. *A **sentential form** of a CFG $\boldsymbol{G}$ is a sequence of nonterminal and terminal symbols $\boldsymbol{\alpha} \in (\mathcal{N} \cup \Sigma)^*$. If $p = (X \to \boldsymbol{\alpha}) \in \mathcal{R}$ and $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2 \in (\mathcal{N} \cup \Sigma)^*$, we write $\boldsymbol{\beta}_1 X \boldsymbol{\beta}_2 \overset{p}{\Longrightarrow} \boldsymbol{\beta}_1 \boldsymbol{\alpha} \boldsymbol{\beta}_2$. If*

$$S \overset{p_1}{\Longrightarrow} \boldsymbol{\alpha}_1 \overset{p_2}{\Longrightarrow} \cdots \overset{p_n}{\Longrightarrow} \boldsymbol{\alpha}_n$$

*where $\boldsymbol{\alpha}_n \in \Sigma^*$, we say that $\boldsymbol{G}$ **derives** $\boldsymbol{\alpha}_n$. The language generated by $\boldsymbol{G}$ is $\mathcal{L}(\boldsymbol{G}) = \{\boldsymbol{s} \mid \boldsymbol{G} \text{ derives } \boldsymbol{s}\}$.*

Definition 29. *The set of **derivation trees** of a CFG is the smallest set defined as follows.*

- *If $a \in \Sigma$ then the tree consisting of a single node a is an a-type derivation tree (whose yield is a).*
- *If $(X \to \beta_1 \cdots \beta_k) \in \mathcal{R}$ and, for $i = 1, \ldots, k$, d_i is a β_i-type derivation tree, then the following is an X-type derivation tree:*



whose yield is $Y(d) = Y(d_1) \cdots Y(d_k)$.

A derivation tree of $\boldsymbol{G}$ is an S-type derivation tree.

A.2 Pushdown Automata

Definition 30. *A **configuration** of a PDA is a pair $(q, \boldsymbol{\gamma})$, where $q \in Q$ is the current state and $\boldsymbol{\gamma}$ is the current contents of the stack.*

Definition 31. *If $(p, X\boldsymbol{\beta})$ and $(q, \boldsymbol{\gamma}\boldsymbol{\beta})$ are configurations, and $\tau = (p, X \xrightarrow{a} q, \boldsymbol{\gamma})$ is a transition, we write $(p, X\boldsymbol{\beta}) \overset{\tau}{\Longrightarrow} (q, \boldsymbol{\gamma}\boldsymbol{\beta})$.*

Definition 32. *A **run** of a PDA is a sequence of transitions $\boldsymbol{\pi} = (\tau_1, \ldots, \tau_n)$ such that there are configurations $(q_0, \boldsymbol{\gamma}_0), \ldots, (q_n, \boldsymbol{\gamma}_n)$ such that*

$$(q_0, \boldsymbol{\gamma}_0) \overset{\tau_1}{\Longrightarrow} \cdots \overset{\tau_n}{\Longrightarrow} (q_n, \boldsymbol{\gamma}_n).$$

If each τ_i scans a_i, then we say that the run scans $a_1 \cdots a_n$.

*A run $\boldsymbol{\pi}$ is called a **pop computation** of X from q_0 to q_n if $\boldsymbol{\gamma}_0 = X\boldsymbol{\gamma}_n$, and for all $i < n$, $|\boldsymbol{\gamma}_i| \geq |\boldsymbol{\gamma}_0|$.*

*An **accepting run** (or **derivation**) is a pop computation of S from q_ι to q_f.*

Definition 33. *Let $\boldsymbol{P}$ be a PDA. If $\boldsymbol{P}$ has an accepting run that scans $\boldsymbol{s} \in \Sigma^*$, we say that $\boldsymbol{P}$ **accepts** $\boldsymbol{s}$. The language **recognized** by $\boldsymbol{P}$ is $\mathcal{L}(\boldsymbol{P}) = \{\boldsymbol{s} \in \Sigma^* \mid \boldsymbol{P} \text{ accepts } \boldsymbol{s}\}$.*

Definition 34. *The set of **derivation trees** of a PDA is the smallest set defined as follows.*

- *The tree consisting of a single node (q_f, ε) (which is the final configuration), is a (q_f, ε)-type derivation tree (whose yield is ε).*
- *If $(q, \boldsymbol{\gamma}) \overset{a}{\Rightarrow} (q', \boldsymbol{\gamma}')$ and D' is a $(q', \boldsymbol{\gamma}')$-type derivation tree, then the following is a $(q, \boldsymbol{\gamma})$-type derivation tree:*

$$D = \begin{array}{c}(q, \boldsymbol{\gamma}) \\ |a \\ D'\end{array}$$

If the yield of D' is w', then the yield of D is aw'.

A derivation tree of P is a (q_ι, S)-type derivation tree, where (q_ι, S) is the initial configuration.

The definitions of runs, accepting runs, and pop computation for LD-PDAs are analogous to those for PDAs.

Proposition 6. *Every PDA is d-strongly equivalent to a PDA with one state.*

Proof. Given a PDA

$$\mathbf{P} = (Q, \Sigma, \Gamma, \delta, (q_\iota, S), (q_f, \varepsilon))$$

construct the PDA

$$\begin{aligned}\mathbf{P}' &= (\{q'\}, \Sigma, \Gamma', \delta', (q', S'), (q', \varepsilon)) \\ \Gamma' &= \{X_{qr} \mid X \in \Gamma, q, r \in Q\} \\ S' &= S^{q_\iota q_f}\end{aligned}$$

and for every transition $(q, X \xrightarrow{a} r, Y_1 Y_2 \cdots Y_k) \in \delta$ with $k > 0$, let δ' contain transitions $(q', X^{q s_k} \xrightarrow{a} q', Y_1^{r s_1} Y_2^{s_1 s_2} \cdots Y_k^{s_{k-1} s_k})$ for all $s_1, \ldots, s_k \in Q$. For every transition $(q, X \xrightarrow{a} r, \varepsilon) \in \delta$, let δ' contain transition $q', X^{qr} \xrightarrow{a} q', \varepsilon$.

We claim that $\mathbf{P}$ and $\mathbf{P}'$ are d-strongly equivalent. ($\Rightarrow$) Given a derivation of $\mathbf{P}$, apply the above construction to each of the transitions used. Although the construction generates many possible sequences of transitions, exactly one sequence is a well-formed derivation. ($\Leftarrow$) Given a derivation of $\mathbf{P}'$, just apply the reverse construction to each of the transitions used. □

B $\mathcal{L}_2$ Formalisms

B.1 Embedded Pushdown Automata

To distinguish between the two types of stacks in an EPDA, we will call them the *outer* stack and the *inner* stacks. A inner stack is written as $[\boldsymbol{\gamma}]$, $\boldsymbol{\gamma} \in \Gamma^*$ is written from top to bottom as in a PDA. We write outer stacks as $\boldsymbol{\Upsilon} = [\boldsymbol{\gamma}_1] \cdots [\boldsymbol{\gamma}_n]$, where $[\boldsymbol{\gamma}_1]$ is at the top, while $[\boldsymbol{\gamma}_n]$ is at the bottom.

Definition 35. *A **configuration** of an EPDA is a pair $(q, \boldsymbol{\Upsilon})$, where q is the current state and $\boldsymbol{\Upsilon}$ is the current contents of the outer stack.*

Definition 36. *If an EPDA has a transition $\tau = (q, [A \cdot\cdot] \xrightarrow{a} r, \boldsymbol{\Upsilon}_1\, [\boldsymbol{\gamma} \cdot\cdot]\, \boldsymbol{\Upsilon}_2)$, then we write, for any $\boldsymbol{\Psi} \in ([\Gamma^*])^*$ and $\boldsymbol{\beta} \in \Gamma^*$,*

$$(q, [A\boldsymbol{\beta}]\, \boldsymbol{\Psi}) \overset{\tau}{\Longrightarrow} (r, \boldsymbol{\Upsilon}_1\, [\boldsymbol{\gamma}\boldsymbol{\beta}]\, \boldsymbol{\Upsilon}_2\, \boldsymbol{\Psi}).$$

Similarly, if an EPDA has a transition $\tau = (q, [A] \xrightarrow{a} r, \varepsilon)$, then we write

$$(q, [A]\, \boldsymbol{\Psi}) \overset{\tau}{\Longrightarrow} (r, \boldsymbol{\Psi}).$$

The definitions of scanning and non-scanning transitions, runs, and accepting runs of an EPDA, and EPDAs accepting strings and recognizing languages, are all analogous to those for PDAs.

Definition 37. *An EPDA is in **normal form** if its transitions have one of the following types*

$$\begin{aligned} p, [A \cdot\cdot] &\xrightarrow{\varepsilon} q, [S] \cdots [\cdot\cdot] \cdots [S], \\ p, [A \cdot\cdot] &\xrightarrow{\varepsilon} p, [B_1 \cdots B_k \cdot\cdot], \\ p, [A] &\xrightarrow{a} q, \varepsilon. \end{aligned}$$

B.2 Linear Indexed Grammars

Definition 38. *A **sentential form** $\boldsymbol{\alpha} \in (\mathcal{N}[\Gamma^*] \cup \Sigma)^*$ of a LIG is a sequence of nonterminal symbols, each augmented with a stack, and terminal symbols.*

Definition 39. *If a LIG has a production $p = (X[A \cdot\cdot] \to \boldsymbol{\Upsilon}_1 Y[\boldsymbol{\gamma} \cdot\cdot]\boldsymbol{\Upsilon}_2)$, then we write, for any $\boldsymbol{\Psi}_1, \boldsymbol{\Psi}_2 \in (\mathcal{N}[\Gamma^*] \cup \Sigma)^*$ and $\boldsymbol{\beta} \in \Gamma^*$,*

$$\boldsymbol{\Psi}_1\, X[A\boldsymbol{\beta}]\, \boldsymbol{\Psi}_2 \overset{p}{\Longrightarrow} \boldsymbol{\Psi}_1\, \boldsymbol{\Upsilon}_1\, Y[\boldsymbol{\gamma}\boldsymbol{\beta}]\, \boldsymbol{\Upsilon}_2\, \boldsymbol{\Psi}_2.$$

Similarly, if a LIG has a production $p = (X[A] \to a)$*, then we write*

$$\Psi_1 X[A] \Psi_2 \overset{p}{\Longrightarrow} \Psi_1 a \Psi_2.$$

A LIG derivation starts with the start symbol and initial stack $S[S]$. The definitions of LIG derivations, LIG derivation trees, LIGs accepting strings, and LIGs recognizing languages are all analogous to those for CFGs.

Definition 40. *A LIG is in* ***normal form*** *if its productions have one of the following types*

$$X[A\,..] \to Y_1[S] \cdots Y_d[..] \cdots Y_k[S],$$
$$X[A\,..] \to X[B_1 \cdots B_k\,..],$$
$$X[A] \to a.$$

B.3 Tree-Adjoining Grammars

Definition 41. *A tree* β *with a foot node is* ***adjoined*** *at an interior node* η *by removing the children of* η*, replacing* η *with* β *and inserting the old children of* η *as the children of the foot node of* β*. The old label of* η *is thus lost.*

A tree α *without a foot node is* ***substituted*** *at a leaf node* η *by replacing* η *with* α*. The old label of* η *is thus lost.*

Definition 42. *If* $\tau = (X \to \beta)$ *is a TAG production,* α *is a tree with a node* η *labeled* X*, and* α' *is the resulting of adjoining or substituting* β *at* η*, we write* $\alpha \overset{\tau}{\Longrightarrow} \alpha'$*.*

Definition 43. *Let* $\boldsymbol{G}$ *be a TAG. If*

$$S \overset{\tau_1}{\Longrightarrow} \alpha_1 \Rightarrow \cdots \overset{\tau_n}{\Longrightarrow} \alpha_n$$

where α_n *does not have any variable nodes, then we say that* $\boldsymbol{G}$ ***derives*** α_n*.*

Definition 44. *The* ***tree language*** *of a TAG is the set of trees it derives. Additionally, the* yield *of a tree is the string obtained by concatenating the terminal symbols at the leaves of the tree, and the* ***string language*** *of a TAG is the set of strings yielded by its tree language.*

Definition 45. *A TAG is in* ***normal form*** *if its productions are of the types shown in Fig. 6.*

B.4 Pushdown Adjoining Automata

Definition 46. *A PAA is in* ***normal form*** *if its transitions are of the types shown in Fig. 7.*

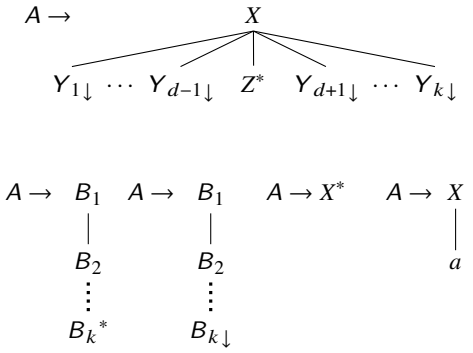


Figure 6: Production rules of TAG in normal form. Symbols X and Z are constant; other uppercase letters are variables.

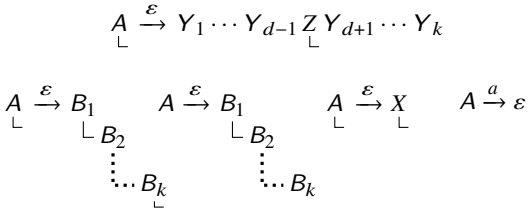


Figure 7: Transitions of PAA in normal form. Symbols X and Z are constant; other uppercase letters are variables.

C Derivation Trees of Two-Level Formalisms

In §5 we showed what the derivation trees of four two-level formalisms look like when both the controller and the controllee are in normal form. In this section, we give details on what the derivation trees look like for general controller CFGs/PDAs and controllee LD-CFGs/LD-PDAs.

PDA ▷ PDA The normal form and general form of PDAs are not so different, so moving to general PDAs is straightforward. We still distinguish between the cases where the controller transition is (a,b) scanning or (c) non-scanning, and if the former, whether the controllee transition has (a) no distinguished symbol or (b) a distinguished symbol.

(a) If a derivation node is labeled $q, X[s, A\boldsymbol{\beta}]\,\Upsilon$, and $s, A \xrightarrow{\ell} q_f, \varepsilon$ is a controller transition, and ℓ is a controllee transition $q, X \xrightarrow{a} r, Y_1 \cdots Y_k$ (with no distinguished symbol),

$$\begin{array}{c} q, X[s, A]\,\Upsilon \\ \Big| a \\ r, Y_1[q_\iota, S_2] \cdots Y_k[q_\iota, S_2]\,\Upsilon \end{array}$$

(b) If $s, A \xrightarrow{\ell} t, \boldsymbol{\gamma}$ is a controller transition, and ℓ is a controllee transition $q, X \xrightarrow{a}$

$r, Y_1 \cdots Y_{d-1}\, \breve{Z}\, Y_{d+1} \cdots Y_k$, then the node can have child

$$\begin{array}{c} q, X[s, A\beta]\,\Upsilon \\ \Big| a \\ r, Y_1[q_\iota, S_2] \cdots Y_{d-1}[q_\iota, S_2]\, Z[t, \gamma\beta]\, Y_{d+1}[q_\iota, S_2] \cdots Y_k[q_\iota, S_2]\,\Upsilon \end{array}$$

Case (c) stays the same.

PDA ▷ CFG Generalizing to a PDA controller is also not very difficult here.

(a) If a derivation node is labeled $X[q, A]$, and $q, A \xrightarrow{\ell} q_f, \varepsilon$ is a controller transition, and ℓ is a controllee production $X \to \upsilon_1 \cdots \upsilon_k$, where each υ_i is either a terminal or non-distinguished nonterminal symbol, then the node can have children as follows. For each i,

- If υ_i is a terminal a, then there is a child a.
- If υ_i is a non-distinguished nonterminal Y, then there is a child $Y[q_\iota, S_2]$.

(b) If a derivation node is labeled $X[q, A\beta]$, and $q, A \xrightarrow{\ell} r, \gamma$ is a controller transition, and ℓ is a controllee production $X \to \upsilon_1 \cdots \upsilon_k$, where each υ_i is either a terminal, nonterminal, or distinguished nonterminal symbol, then the node can have children as follows. For each i,

- If υ_i is a terminal or non-distinguished nonterminal, then there is a child as in case (a).
- If υ_i is a distinguished nonterminal Z, then there is a child $Z[r, \gamma\beta]$.

Case (c) stays the same.

CFG ▷ CFG Generalizing to a CFG controller is more complicated because a production can now yield multiple terminal symbols, which causes the controllee to use multiple productions in a single derivation step.

Derivation nodes are labeled ${}^{X}\!/_{Z}[A]$ where A is a controller nonterminal, and X and Z are controllee nonterminals or $\bot$. We write ${}^{X}\!/_{\bot}[A]$ as $X[A]$.

If a derivation node is labeled ${}^{X}\!/_{Z}[A]$, and $A \to \beta_1 \cdots \beta_k$ is a controller production, where each β_i is either a nonterminal or a terminal symbol, then the derivation node can have children as follows. Let $Y_0 = X$, $Y_k = Z$, and $Y_1, \ldots, Y_{k-1}$ be any controllee nonterminals. For each i:

(a) If β_i is a terminal ℓ that names a controllee production $Y_{i-1} \to \upsilon_1 \cdots \upsilon_m$, where each υ_j is a terminal or non-distinguished nonterminal (and $Y_i = \bot$), then for each j,

- If υ_j is a terminal a, then there is a child a.
- If υ_j is a non-distinguished nonterminal Y, then there is a child $Y[S_2]$.

(b) If β_i is a terminal ℓ that names a controllee production $Y_{i-1} \to \upsilon_1 \cdots \upsilon_m$, where each υ_j is a terminal, nonterminal or distinguished nonterminal (and $Y_i \neq \bot$), then for each j,

- If υ_j is a terminal or non-distinguished nonterminal, then there is a child as in case (a).
- If υ_j is a distinguished nonterminal, it must be $\breve{Y}_i$ (and $Y_i \neq \bot$).

(c) If β_i is a nonterminal B, then there is a child ${}^{Y_{i-1}}\!/_{Y_i}[B]$.

CFG ▷ PDA Derivation nodes are labeled (q, Υ), where q is a state and Υ is a string of symbols of the form ${}^{X}\!/_{Z}[A]$ (as above, for CFG ▷ CFG) or $*$.

If a derivation node is labeled $q, *^{?}{}^{X}\!/_{Z}[A]\,\Upsilon$ (where $*^{?}$ denotes either the presence or absence of a $*$), and $A \to \beta_1 \cdots \beta_k$ is a controller production, where each β_i is either a nonterminal or terminal symbol, then the derivation node can have children as follows. Let $Y_0 = X$, $Y_k = Z$, and $Y_1, \ldots, Y_{k-1}$ be any controllee nonterminals. For each i:

(a) If β_i is a terminal ℓ that names a controllee transition $q, Y_{i-1} \xrightarrow{a} q, W_1 \cdots W_k$ (with no distinguished symbol and $Y_i = \bot$), then there is a child

$$\begin{array}{c} \Big| a \\ q, W_1[S_2] \cdots W_k[S_2] \end{array}$$

(b) If β_i is a terminal ℓ that names a controllee transition $q, Y_{i-1} \xrightarrow{a} q, W_1 \cdots W_{d-1}\, \breve{Y}_i\, W_{d+1} \cdots W_k$ (and $Y_i \neq \bot$), then there is a child

$$\begin{array}{c} \Big| a \\ q, W_1[S_2] \cdots W_{d-1}[S_2] * W_{d+1}[S_2] \cdots W_k[S_2] \end{array}$$

(c) If β_i is a nonterminal B, then there is a child

$$\begin{array}{c} \Big| \varepsilon \\ q, {}^{Y_{i-1}}\!/_{Y_i}[B] \end{array}$$

Finally, append Υ to the leftmost child.

D D-Weak Equivalence Proofs

Proposition 7. *CFG and PDA are d-weakly equivalent.*

Proof. ($\Rightarrow$) Let $\mathbf{G} = (\mathcal{N}, \Sigma, \mathcal{R}, S)$ be a CFG in normal form. We define the PDA in normal form $\mathbf{P_G} = (\{q\}, \Sigma, \mathcal{N}, \delta, (q, S), (q, \varepsilon))$, where

$$\delta = \{q, A \xrightarrow{\varepsilon} q, B_1 \cdots B_k \mid A \to B_1 \cdots B_k \in \mathcal{R}\} \cup \{q, A \xrightarrow{a} q, \varepsilon \mid A \to a \in \mathcal{R}\}.$$

We prove by induction on the height h of the derivation tree that for each A-type derivation tree d of $\mathbf{G}$ that yields w, $\mathbf{P_G}$ has a pop computation of A that scans w.

Base case: When the height is $h = 2$, it must hold that $\mathbf{G}$ has a production $A \to a$. The yield of the derivation is a. By construction, $\mathbf{P_G}$ has the transition $q, A \xrightarrow{a} q, \varepsilon$, which is a pop computation of A that scans a.

Inductive Step: We assume that the statement holds for each A-type derivation of height up to h and we assume that d is a derivation tree of height $h + 1$ yielding $w = w_1 w_2 \cdots w_k$, which has root A and children $B_1, B_2, \ldots, B_k$. We assume that the $B_1, B_2, \ldots, B_k$-type derivations have yield $w_1, w_2, \ldots, w_k$. This implies that $\mathbf{G}$ has the production $A \to B_1 B_2 \cdots B_k$ and $\mathbf{P_G}$ has the transition $q, A \xrightarrow{\varepsilon} q, B_1 B_2 \cdots B_k$. By the inductive hypothesis, $\mathbf{P_G}$ has pop computations (from state q to state q) of $B_1, B_2, \ldots, B_k$ scanning $w_1, w_2, \ldots, w_k$, thus it has a pop computation of A that yields w.

Therefore, there is a yield-preserving bijection between the S-type derivations of $\mathbf{G}$ and $\mathbf{P_G}$'s pop computations of S (accepting runs).

($\Leftarrow$) Let $\mathbf{P} = (Q, \Sigma, \Gamma, (q_\iota, S), (q_f, \varepsilon))$ be a PDA in normal form. We define the CFG in normal form

$$\begin{aligned} \mathbf{G_P} &= (\mathcal{N}, \Sigma, \mathcal{R}, S') \\ \mathcal{N} &= \{A^{pq} \mid A \in \Gamma, p, q \in Q\} \\ S' &= S^{q_\iota q_f} \end{aligned}$$

and $\mathcal{R}$ is constructed as follows:

- For each transition $(p, A \xrightarrow{\varepsilon} q, B_1 \cdots B_k) \in \delta$, and for each $r_1, r_2, \ldots, r_k \in Q$, include production $A_{pr_k} \to B_1^{qr_1} B_2^{r_1 r_2} \cdots B_k^{r_{k-1} r_k}$.
- For each transition $(p, A \xrightarrow{a} q, \varepsilon) \in \delta$, include production $A^{pq} \to a$.

We prove that for each pop computation of A of $\mathbf{P}$ from p to q that scans w, there is a A^{pq}-type derivation of $\mathbf{G_P}$ with yield w. We prove the statement by induction on the length l of the pop computation.

Base case: When the pop computation has length $l = 1$, it must consist of a single transition $p, S \xrightarrow{a} q, \varepsilon$. It scans the string a. By construction, $\mathbf{G_P}$ has a production $S^{pq} \to a$, thus it has a derivation yielding a.

Inductive Step: We assume that the statement holds for all pop computations of length up to l and $\boldsymbol{\pi}$ is a pop computation of A such that its first transition is $p, A \xrightarrow{\varepsilon} q, B_1 B_2 \cdots B_k$, followed by pop computations of $B_1, B_2, \ldots, B_k$ from p to r_1, r_1 to r_2, ..., and r_{k-1} to r_k, respectively. We assume that the pop computations of $B_1, B_2, \ldots, B_k$ scan $w_1, w_2, \ldots, w_k$, therefore $\boldsymbol{\pi}$ scans $w = w_1 w_2 \cdots w_k$. By construction, $\mathbf{G_P}$ has a production $A^{pq} \to B_1^{pr_1} B_2^{r_1 r_2} \cdots B_k^{r_{k-1} r_k}$. By the inductive hypothesis $\mathbf{G_P}$ has $B_1^{pr_1} B_2^{r_1 r_2} \cdots B_k^{r_{k-1} r_k}$-type derivations yielding $w_1 w_2 \cdots w_k$, thus it also has a A^{pr_k}-type derivation yielding w.

Therefore, there is a yield-preserving bijection between the accepting runs of $\mathbf{P}$ (pop computations of S from q_ι to q_f) and derivation trees of $\mathbf{G_P}$. □

Proposition 8. *LD-CFG and LD-PDA are d-weakly equivalent, in the sense that for any LD-CFG $\boldsymbol{G}$, there is an LD-PDA $\boldsymbol{P_G}$ such that for any controller $\boldsymbol{F}$, $\boldsymbol{F} \triangleright \boldsymbol{G}$ and $\boldsymbol{F} \triangleright \boldsymbol{P_G}$ are d-weakly equivalent; conversely, for any any LD-PDA $\boldsymbol{P}$, there is an LD-PDA $\boldsymbol{G_P}$ such that for any controller $\boldsymbol{F}$, $\boldsymbol{F} \triangleright \boldsymbol{P}$ and $\boldsymbol{F} \triangleright \boldsymbol{G_P}$ are d-weakly equivalent.*

Proof. ($\Rightarrow$) Let $\mathbf{G} = (\mathcal{N}, \Sigma, L, \mathcal{R}, S)$ be an LD-CFG in normal form. We define the LD-PDA

$$\mathbf{P_G} = (\{q\}, \Sigma, \mathcal{N}, L, \delta, (q, S), (q, \varepsilon))$$

where δ is constructed as follows:

- For each $(\ell\colon A \to B_1 \cdots \breve{B}_d \cdots B_k)$ in $\mathcal{R}$, include transition $\ell\colon q, A \xrightarrow{\varepsilon} q, B_1 \cdots \breve{B}_d \cdots B_k$.
- For each $(\ell\colon A \to a) \in \mathcal{R}$, include transition $\ell\colon q, A \xrightarrow{a} q, \varepsilon$.

The rest of the proof is similar to the proof of Prop. 7. The only addition is to observe that $\mathbf{P_G}$ uses each control word the same number of times that $\mathbf{G}$ does, ensuring that the derivations of $\mathbf{F}$,

$\mathbf{F} \triangleright \mathbf{G}$ and $\mathbf{F} \triangleright \mathbf{P_G}$ are in one-to-one correspondence.

($\Leftarrow$) Let $\mathbf{P} = \big(Q, \Sigma, \Gamma, L, (q_\iota, S), \big(q_f, \varepsilon\big)\big)$ be an LD-PDA in normal form. We define the LD-CFG in normal form

$$\begin{aligned}\mathbf{G_P} &= (\mathcal{N}, \Sigma, \mathcal{R}, S')\\ \mathcal{N} &= \{A^{pq} \mid A \in \Gamma, p, q \in Q\}\\ S' &= S^{q_\iota q_f}\end{aligned}$$

and $\mathcal{R}$ is constructed as follows:

- For each $\ell\colon p, A \xrightarrow{\varepsilon} q, B_1 \cdots \breve{B}_d \cdots B_k$ in δ, and for each $r_1, r_2, \ldots, r_k \in Q$, include production $\ell\colon A^{pr_k} \to B_1^{qr_1} B_2^{r_1 r_2} \cdots \breve{B}_d^{r_{d-1} r_d} \cdots B_k^{r_{k-1} r_k}$.
- For each transition $(\ell\colon p, A \xrightarrow{a} q, \varepsilon) \in \delta$, include production $\ell\colon A^{pq} \to a$.

Again, the rest of the proof is similar to the proof of Prop. 7. □

Proposition 1. *CFG $\triangleright$ CFG, PDA $\triangleright$ CFG, CFG $\triangleright$ PDA and PDA $\triangleright$ PDA are d-weakly equivalent.*

Proof. When two two-level formalisms have the same controllee but one has a controller CFG and the other has a controller PDA, it is sufficient to define controllers that are d-weakly equivalent. The proof of Prop. 7 shows that this is possible and how the controllers can be defined. If there is a bijection between the derivations of the CFG and PDA for each control word, the rules of the controllee will be applied in the same order in both two-level formalisms, thus generating the same set of strings with an equal number of derivations.

When two two-level formalisms have the same controller but different controllees, the controllees must consume the control words in the same order and the same number of times. Indeed, the proof of Prop. 8 shows that this is possible and how the controllees can be defined.

□

E D-Strong Equivalence Proofs

We assume for simplicity that the controller CFGs/PDAs and the LD-CFGs/LD-PDAs are in normal form in this section. However, the proofs of equivalence also hold in the general case.

Proposition 2. *EPDA and PDA $\triangleright$ PDA are d-strongly equivalent.*

Proof. ($\Rightarrow$) Let

$$\mathbf{P} = \big(Q, \Sigma, \Gamma, \delta, (q_\iota, [S]), (q_f, \varepsilon)\big)$$

be an EPDA in normal form. Construct the controllee LD-PDA and controller PDA

$$\begin{aligned}\mathbf{P}_1 &= (Q, \Sigma, \{X\}, L, \delta_1, (q_\iota, X), (q_f, \varepsilon))\\ \mathbf{P}_2 &= (\{q\}, L, \Gamma, \delta_2, (q, S), (q, \varepsilon))\end{aligned}$$

where the set of labels L and the sets of transitions δ_1 and δ_2 are constructed as follows:

- For each transition $(p, [A \cdot\cdot] \xrightarrow{\varepsilon} p, [\gamma \cdot\cdot]) \in \delta$, let L contain a fresh label ℓ, let δ_2 contain the transition $q, A \xrightarrow{\ell} q, \gamma$, and let δ_1 contain the transition $\ell\colon p, X \xrightarrow{\varepsilon} p, X$.
- For each transition $(p, [A \cdot\cdot] \xrightarrow{\varepsilon} r, [S] \ldots [\cdot\cdot] \ldots [S])$ in δ, add a new label ℓ to L, the transition $q, A \xrightarrow{\ell} q, \varepsilon$ to δ_2 and the transition $\ell\colon p, X \xrightarrow{\varepsilon} r, X \cdots \breve{X} \cdots X$ to δ_1.
- For each transition $(p, [A] \xrightarrow{a} r, \varepsilon) \in \delta$, add a new label ℓ to L, the transition $q, A \xrightarrow{\ell} q, \varepsilon$ to δ_2 and the transition $\ell\colon p, X \xrightarrow{a} r, \varepsilon$ to δ_1.

($\Leftarrow$) We are given $\mathbf{P}_2 \triangleright \mathbf{P}_1$, and by Prop. 6 we can assume without loss of generality that both $\mathbf{P}_1$ and $\mathbf{P}_2$ have only one state. Thus

$$\begin{aligned}\mathbf{P}_1 &= (\{q\}, \Sigma, \Gamma_1, L, \delta, (q, S_1), (q, \varepsilon))\\ \mathbf{P}_2 &= (\{q\}, L, \Gamma_2, \delta_2, (q, S_2), (q, \varepsilon))\end{aligned}$$

and we construct the EPDA

$$\begin{aligned}\mathbf{P}' &= (\{q\}, \Sigma, \Gamma', \delta', (q, [S_1[S_2]]), (q, \varepsilon))\\ \Gamma' &= \{{}^{X}/_{Y}[A] \mid A \in \Gamma_2, X, Y \in \Gamma_1\}\\ &\quad \cup \{X[A] \mid A \in \Gamma_2, X \in \Gamma_1\}.\end{aligned}$$

The set δ' is constructed as follows:

- For each transition $(q, A \xrightarrow{\varepsilon} q, B_1 \cdots B_k) \in \delta_2$ where $k > 0$, and for each $X_0, \ldots, X_k \in \Gamma_1$, let δ' contain the transition $q, [{}^{X_0}/_{X_k}[A] \cdot\cdot] \xrightarrow{\varepsilon} q, [{}^{X_0}/_{X_1}[B_1] \cdots {}^{X_{k-1}}/_{X_k}[B_k] \cdot\cdot]$.
- For each transition $(q, A \xrightarrow{\varepsilon} q, \varepsilon) \in \delta_2$, and for each $X \in \Gamma_1$, let δ' contain the transition $q, [{}^{X}/_{X}[A] \cdot\cdot] \xrightarrow{\varepsilon} q, [\cdot\cdot]$.

- For each pair of transitions $(q, A \xrightarrow{\ell} q, \varepsilon) \in \delta_2$ and $(\ell\colon q, X \xrightarrow{\varepsilon} q, Y_1 \cdots \check{Y}_d \cdots Y_k) \in \delta_1$, let δ' contain the transition $q, [{}^{X}/_{Y_d}[A]\,\cdot\cdot] \xrightarrow{\varepsilon} q, [Y_1[S_2]] \cdots [\cdot\cdot] \cdots [Y_k[S_2]]$ (where the $[\cdot\cdot]$ is in the d-th position).

- For each pair of transitions $(q, A \xrightarrow{\ell} q, \varepsilon) \in \delta_2$ and $(\ell\colon q, X \xrightarrow{a} q, \varepsilon) \in \delta_1$, let δ' contain the transition $q, [X[A]] \xrightarrow{a} q, \varepsilon$. □

Proposition 3. *LIG and PDA ▸ CFG are d-strongly equivalent.*

Proof. ($\Rightarrow$) Let $\mathbf{G} = (\mathcal{N}, \Sigma, \Gamma, \mathcal{R}, S, S)$ be a LIG in normal form. Construct the controllee LD-CFG and controller PDA

$$\mathbf{G}_1 = (\mathcal{N}, \Sigma, L, \mathcal{R}_1, S)$$
$$\mathbf{P}_2 = (\{q\}, L, \Gamma, \delta_2, (q, S), (q, \varepsilon))$$

The set of labels L, the set of productions $\mathcal{R}_1$ and the set of transitions δ_2 are constructed as follows:

- For each production $X[A\,\cdot\cdot] \to X[\boldsymbol{\gamma}\,\cdot\cdot] \in \mathcal{R}$, δ_2 contains the transition $q, A \xrightarrow{\varepsilon} q, \boldsymbol{\gamma}$.

- For each production $X[A\,\cdot\cdot] \to Y_1[S] \cdots Y_d[\,\cdot\cdot] \cdots Y_k[S]$, add a fresh label ℓ to L, the transition $q, A \xrightarrow{\ell} q, \varepsilon$ to δ_2 and the production $\ell\colon X \to Y_1 \cdots \check{Y}_d \cdots Y_k$ to $\mathcal{R}_1$.

- For each production $X[A] \to a$, add a fresh label ℓ to L, the transition $q, A \xrightarrow{\ell} q, \varepsilon$ to δ_2 and the production $\ell\colon X \to a$ to $\mathcal{R}_1$.

($\Leftarrow$) Let $\mathbf{G}_1 = (\mathcal{N}, \Sigma, L, \mathcal{R}_1, S)$ be a controllee LD-CFG and $\mathbf{P}_2 = \big(Q, L, \Gamma, \delta_2, (q_\iota, S), (q_f, \varepsilon)\big)$ a controller PDA, both in normal form. Construct the LIG

$$\mathbf{G}' = (\mathcal{N}', \Sigma, \Gamma, \mathcal{R}', S, S)$$
$$\mathcal{N}' = \{X^q \mid X \in \mathcal{N}, q \in Q\}$$

and the set $\mathcal{R}'$ is defined as follows:

- For each transition $(p, A \xrightarrow{\varepsilon} q, \boldsymbol{\gamma}) \in \delta_2$, and for each $X \in \mathcal{N}$, let $\mathcal{R}'$ contain a production $X^p[A\,\cdot\cdot] \to X^q[\boldsymbol{\gamma}\,\cdot\cdot]$.

- For each transition $(p, A \xrightarrow{\ell} q, \varepsilon) \in \delta_2$ and production $(\ell\colon X \to Y_1 \cdots \check{Y}_d \cdots Y_k) \in \mathcal{R}_1$, let $\mathcal{R}'$ contain a production $X^p[A\,\cdot\cdot] \to Y_1^{q_\iota}[S] \cdots Y_d^{q}[\,\cdot\cdot] \cdots Y_k^{q_\iota}[S]$.

- For each transition $(p, A \xrightarrow{\ell} q_f, \varepsilon) \in \delta_2$ and production $(\ell\colon X \to a) \in \mathcal{R}_1$, let $\mathcal{R}'$ contain a production $X^p[A] \to a$. □

Proposition 4. *Spinal TAG and CFG ▸ CFG are d-strongly equivalent.*

Proof. ($\Rightarrow$) Let $\mathbf{T} = (\mathcal{V}, C, \Sigma, S, \mathcal{R})$ be a spinal TAG in normal form.

First, rename apart variables that allow adjunction and variables that allow substitution.

Then construct the controllee LD-CFG and controller CFG

$$\mathbf{G}_1 = (C \cup \{\square\}, \Sigma, L, \mathcal{R}_1, S_1)$$
$$\mathbf{G}_2 = (\mathcal{V}, L, \mathcal{R}_2, S)$$

where $\square \notin C$ is a fresh constant symbol, and the set of labels L and the sets of rules $\mathcal{R}_1$ and $\mathcal{R}_2$ are constructed as follows:

- For each rule of the type

$$\begin{array}{rl} A \to & B_1 \\ & \vert \\ & B_2 \\ & \vdots \\ & B_k{}^* \end{array}$$

  $\mathcal{R}_2$ contains the production $A \to B_1 B_2 \cdots B_k$.

- For each rule of the type

$$\begin{array}{rl} A \to & B_1 \\ & \vert \\ & B_2 \\ & \vdots \\ & B_{k\downarrow} \end{array}$$

  L contains fresh labels ℓ_0 and ℓ_k, $\mathcal{R}_2$ contains the production $S \to \ell_0 B_1 B_2 \cdots B_{k-1} \ell_k S$, and $\mathcal{R}_1$ contains the productions

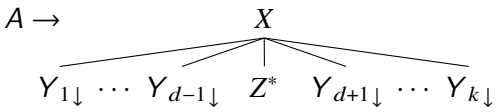
$$\ell_0\colon A \to \square$$
$$\ell_k\colon \square \to B_k$$

- For each rule of the type

$$A \to X^*$$

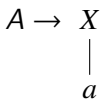
  $\mathcal{R}_2$ contains the production $A \to \varepsilon$.

- For each rule of the type



L contains a fresh symbol ℓ, $\mathcal{R}_1$ contains the production $\ell\colon \square \to Y_1 \cdots Y_{d-1} \breve{\square} Y_{d+1} \cdots Y_k$ and $\mathcal{R}_2$ contains the production $A \to \ell$.

- For each rule of the type



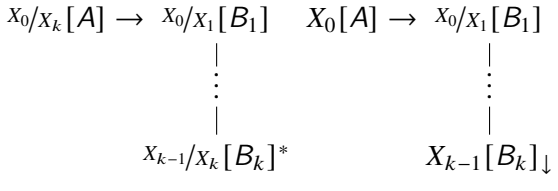
L contains a fresh symbol ℓ, $\mathcal{R}_1$ contains the production $\ell\colon A \to a$ and $\mathcal{R}_2$ contains the production $S \to \ell$.

($\Leftarrow$) Let $\mathbf{G}_1 = (\mathcal{N}_1, \Sigma, L, \mathcal{R}_1, S_1)$ be an LD-CFG and $\mathbf{G}_2 = (\mathcal{N}_2, L, \mathcal{R}_2, S_2)$ a controller CFG. Construct the TAG

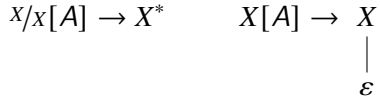
$$\begin{aligned}\mathbf{T} &= (\mathcal{V}, \mathcal{N}, \Sigma, S_1[S_2], \mathcal{R})\\ \mathcal{V} &= \{{}^{X}/_{Y}[A] \mid X, Y \in \mathcal{N}, A \in \mathcal{N}'\}\\ &\quad \cup \{X[A] \mid X \in \mathcal{N}, A \in \mathcal{N}'\}\end{aligned}$$

and the set $\mathcal{R}$ is constructed as follows:

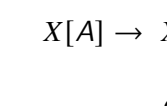
- For each production $A \to B_1 \cdots B_k \in \mathcal{R}_2$, where $k > 0$, and for each $X_0, X_1, \ldots, X_k \in \mathcal{N}$, let $\mathcal{R}$ contain the rules



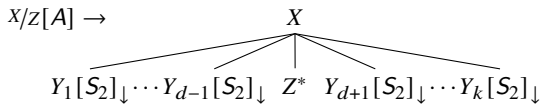
- For each production $A \to \varepsilon \in \mathcal{R}_2$, let $\mathcal{R}$ contain the rules



- For each pair of productions $A \to \ell \in \mathcal{R}_2$ and $\ell\colon X \to a \in \mathcal{R}_1$, let $\mathcal{R}$ contain the rule



- For each pair of productions $A \to \ell \in \mathcal{R}_2$ and $\ell\colon X \to Y_1 \cdots Y_{d-1} \breve{Z} Y_{d+1} \cdots Y_k \in \mathcal{R}_1$, let $\mathcal{R}$ contain the rule



□

Proposition 5. *Spinal PAA and CFG ▷ PDA are d-strongly equivalent.*

Proof. ($\Rightarrow$) Let $\mathbf{M} = (\Sigma, \mathcal{V}, C, \delta, S)$ be a PAA in normal form.

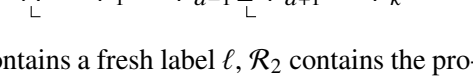
First, rename apart variables that allow adjunction and variables that allow substitution.

Then construct the LD-PDA controllee and CFG controller

$$\begin{aligned}\mathbf{P}_1 &= (\{q\}, \Sigma, C \cup \{\square\}, L, \delta_1, S_1)\\ \mathbf{G}_2 &= (\mathcal{V}, L, \mathcal{R}_2, S)\end{aligned}$$

where $\square \notin C$ is a fresh constant symbol, and the sets L, δ_1 and $\mathcal{R}_2$ are constructed as follows:

- For every transition in δ of the form

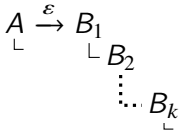


L contains a fresh label ℓ, $\mathcal{R}_2$ contains the production $A \to \ell$ and δ_1 contains the transition $\ell\colon q, \square \xrightarrow{\varepsilon} q, Y_1 \cdots Y_{d-1} \breve{\square} Y_{d+1} \cdots Y_k$.

- For every transition in δ of the form

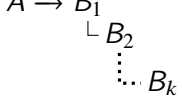
L contains a fresh label ℓ, $\mathcal{R}_2$ contains the production $S \to \ell$ and δ_1 contains the transition $\ell\colon q, A \xrightarrow{a} q, \varepsilon$.

- For every transition in δ of the form



$\mathcal{R}_2$ contains the production $A \to B_1 \cdots B_k$.

- For every transition in δ of the form



L contains fresh labels ℓ_0 and ℓ_k, $\mathcal{R}_2$ contains the production $S \to \ell_0 B_1 \cdots B_{k-1} \ell_k S$ and δ_1 contains the transitions

$$\ell_0 \colon q, A \xrightarrow{\varepsilon} q, \square$$
$$\ell_k \colon q, \square \xrightarrow{\varepsilon} q, B_k$$

- For every transition in δ of the form

$$\underline{A} \xrightarrow{\varepsilon} \underline{X}$$

  $\mathcal{R}_2$ contains the production $A \to \varepsilon$.

($\Leftarrow$) We are given $\mathbf{G}_2 \triangleright \mathbf{P}_1$, where $\mathbf{G}_2$ is a CFG and $\mathbf{P}_1$ is an LD-PDA. Without loss of generality, we may assume by Prop. 6 that $\mathbf{P}_1$ has only one state, so

$$\mathbf{P}_1 = (\{q\}, \Sigma, \Gamma, L, \delta_1, (q_\iota, S_1), (q_f, \varepsilon))$$
$$\mathbf{G}_2 = (\mathcal{N}, L, \mathcal{R}_2, S_2)$$

We further assume that both are in normal form. Then we construct the PAA

$$\mathbf{M} = (\Sigma, \mathcal{V}, \Gamma, \delta', S_1[S_2])$$
$$\mathcal{V} = \{{}^{X}/_{Y}[A] \mid A \in \mathcal{N}, X, Y \in \Gamma\}$$
$$\cup \{X[A] \mid A \in \mathcal{N}, X \in \Gamma\}$$

where δ' is constructed as follows:

- For each production $A \to B_1 \cdots B_k \in \mathcal{R}_2$, where $k > 0$, and for all $X_0, X_1, \ldots, X_k \in \Gamma$, let δ' contain a transition

$$\underline{{}^{X_0}/_{X_k}[A]} \xrightarrow{\varepsilon} {}^{X_0}/_{X_1}[B_1]$$
$$\quad \vdots$$
$$\quad \underline{{}^{X_{k-1}}/_{X_k}[B_k]}$$

- For each production $A \to \varepsilon \in \mathcal{R}_2$ and for all $X \in \Gamma$, let δ' contain a transition

$$\underline{{}^{X}/_{X}[A]} \xrightarrow{\varepsilon} \underline{X}$$

- For all pairs of rules $A \to \ell \in \mathcal{R}_2$ and $\ell \colon q, X \xrightarrow{a} q, \varepsilon \in \delta_1$, let δ' contain the transition $X[A] \xrightarrow{a} \varepsilon$.

- For all pairs of rules $A \to \ell \in \mathcal{R}_2$ and $\ell \colon q, X \xrightarrow{\varepsilon} q, Y_1 \cdots Y_{d-1} \check{Z} Y_{d+1} \cdots Y_k \in \delta_1$, let δ' contain a transition

$$\underline{{}^{X}/_{Z}[A]} \xrightarrow{\varepsilon} Y_1[S_2] \cdots Y_{d-1}[S_2] \underline{Z} Y_{d+1}[S_2] \cdots Y_k[S_2]$$

□

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitations (not numbered)

☑ A2. Did you discuss any potential risks of your work?
Ethics Statement (not numbered)

☑ A3. Do the abstract and introduction summarize the paper's main claims?
Abstract (not numbered) & Introduction (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?

Left blank.

☐ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
No response.

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?
No response.

☐ 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?
No response.

☐ 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.)?
No response.

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.