Probabilistic Parsing Strategies*

Mark-Jan Nederhof
Faculty of Arts
University of Groningen
P.O. Box 716
NL-9700 AS Groningen, The Netherlands
markjan@let.rug.nl

Giorgio Satta
Department of Information Engineering
University of Padua
via Gradenigo, 6/A
I-35131 Padova, Italy
satta@dei.unipd.it

Abstract

We present new results on the relation between purely symbolic context-free parsing strategies and their probabilistic counter-parts. Such parsing strategies are seen as constructions of push-down devices from grammars. We show that preservation of probability distribution is possible under two conditions, viz. the correct-prefix property and the property of strong predictiveness. These results generalize existing results in the literature that were obtained by considering parsing strategies in isolation. From our general results we also derive negative results on (nondeterministic) LR parsing.

Keywords: parsing algorithms, probabilistic parsing, context-free grammars, push-down automata, transduction

1 Introduction

Context-free grammars and push-down automata are two equivalent formalisms to describe context-free languages. While a context-free grammar (CFG) can be thought of as a purely declarative specification, a push-down automaton (PDA) is considered to be an operational specification that determines which steps are performed for a given string in the process of deciding its membership of the language. By a *parsing strategy* we mean a mapping from CFGs to equivalent PDAs, such that some specific conditions are observed. Examples of parsing strategies are top-down parsing (Harrison (1978)), left-corner parsing (Rosenkrantz and Lewis II (1970)), and (nondeterministic) LR parsing (Sippu and Soisalon-Soininen (1990)).

This paper deals with the probabilistic extensions of CFGs and PDAs, i.e., probabilistic CFGs (PCFGs) (Booth and Thompson (1973)) and probabilistic PDAs (PPDAs) (Santos (1972); Tendeau (1995); Abney et al. (1999)). These formalisms are obtained by adding probabilities to the rules and transitions, respectively.

For a given CFG, a particular mapping from rules to probabilities describes a probability distribution over the set of derivations. This is useful for disambiguation of input (Manning and Schütze (1999)). Alternatively, one may construct a PDA from the CFG, according to some parsing strategy, together with a mapping from computations of this PDA to derivations of the

*We gratefully acknowledge very helpful correspondence with David McAllester, Giovanni Pighizzini, Detlef Prescher, Virach Sornlertlamvanich and Eric Villemonte de la Clergerie.

CFG. A particular mapping from transitions to probabilities then indirectly describes a probability distribution over the set of derivations of the CFG.

The mappings from rules or transitions to probabilities cannot be chosen entirely arbitrarily. For example, for a given nonterminal, the sum of probabilities of all rules defining this nonterminal must be 1. This means that only for the first $m-1$ rules out of a total of m rules defining the nonterminal, the probabilities are 'free' parameters, but the m-th is not since it is uniquely determined by the first $m-1$ parameters.

Depending on the choice of the parsing strategy, the constructed PDA may allow different probability distributions than the CFG, since the set of free parameters may differ between CFG and PDA, both quantitatively and qualitatively. For example, Sornlertlamvanich et al. (1999) and Roark and Johnson (1999) have shown that a probability distribution that can be obtained by training the probabilities of a CFG on the basis of a corpus can be less accurate than that obtained by training the probabilities of a PDA constructed by a particular parsing strategy, on the basis of the same corpus. Also the results from Chitrao and Grishman (1990) and Charniak and Carroll (1994) could be seen in this light.

What has been lacking however is a theoretical framework to relate the parameter space of a CFG to that of a PDA constructed from the CFG by a particular parsing strategy, in terms of the set of allowable probability distributions over derivations. Note that the number of free parameters alone is not a satisfactory characterization of the parameter space. In fact, if the 'nature' of the parameters is ill-chosen, then an increase in the number of parameters may lead to a deterioration of the accuracy of the model, due to sparseness of data.

In this paper we investigate conditions on parsing strategies that determine whether all probability distributions allowed by a CFG are also allowed by the PDA constructed from the CFG. For this to be true, the number of free parameters of the PDA must be at least as large as the number of free parameters of the CFG, but the 'nature' of the free parameters of the PDA must also be such that the entire parameter space of the CFG can be covered.

Formulated in different terms, we investigate which parsing strategies can be 'extended' to become *probabilistic* parsing strategies. These are mappings from PCFGs to PPDAs that preserve the induced probability distributions. Two properties of parsing strategies are of particular interest, viz., the correct-prefix property (CPP) and the strong predictiveness property (SPP), both to be defined in the sequel. The main results presented in this paper can be stated as follows:

- No parsing strategy that lacks the correct-prefix property can be extended to become a probabilistic parsing strategy.
- All parsing strategies that possess the correct-prefix property and the strong predictiveness property can be extended to become probabilistic parsing strategies.

The above results generalize previous findings reported by Tendeau (1995), Tendeau (1997) and Abney et al. (1999), where only a few specific parsing strategies were considered in isolation. Our findings also have important implications for well-known parsing strategies such as nondeterministic LR parsing, henceforth simply called 'LR parsing'.[1] LR parsing has the CPP, but lacks the SPP, and as we will show, LR parsing cannot be extended to become a probabilistic parsing strategy. In fact, there are cases where LR parsers have fewer free parameters than the CFGs from which they were constructed, contrary to what is commonly believed.

The paper is organized as follows. After giving standard definitions in Section 2, we give our formal definition of 'parsing strategy' in Section 3. We also define what it means to extend a parsing strategy to become a probabilistic parsing strategy. The CPP and the SPP are defined in Sections 4 and 5, where we also discuss how these properties relate to the question of which strategies can be extended to become probabilistic. Section 6 provides examples of parsing strategies with and without the SPP, and explains why the LR parsing strategy cannot be extended to become probabilistic. We end this paper with conclusions.

[1] Generalized (or nondeterministic) LR parsing allows for more than one action for a given LR state and input symbol.

2 Preliminaries

A context-free grammar (CFG) $\mathcal{G}$ is a 4-tuple (Σ, N, S, R), where Σ is a finite set of *terminals*, called the *alphabet*, N is a finite set of *nonterminals*, including the *start symbol* S, and R is a finite set of *rules*, each of the form $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (\Sigma \cup N)^*$.

For a fixed CFG $\mathcal{G}$, we define the relation $\Rightarrow$ on triples consisting of two strings $\alpha, \beta \in (\Sigma \cup N)^*$ and a rule $\pi \in R$ by: $\alpha \stackrel{\pi}{\Rightarrow} \beta$ if and only if α is of the form $wA\delta$ and β is of the form $w\gamma\delta$, for some $w \in \Sigma^*$ and $\delta \in (\Sigma \cup N)^*$, and $\pi = (A \rightarrow \gamma)$. A *left-most derivation* is a string $d = \pi_1 \cdots \pi_m$, $m \geq 0$, such that $S \stackrel{\pi_1}{\Rightarrow} \cdots \stackrel{\pi_m}{\Rightarrow} \alpha$, for some $\alpha \in (\Sigma \cup N)^*$. We will identify a left-most derivation with the sequence of strings over $\Sigma \cup N$ that arise in that derivation. In the remainder of this paper, we will let the term 'derivation' refer to 'left-most derivation', unless specified otherwise.

A derivation $d = \pi_1 \cdots \pi_m$, $m \geq 0$, such that $S \stackrel{\pi_1}{\Rightarrow} \cdots \stackrel{\pi_m}{\Rightarrow} w$ where $w \in \Sigma^*$ will be called a *complete* derivation; we also say that d is a derivation of w. By *subderivation* we mean a substring of a complete derivation of the form $d = \pi_1 \cdots \pi_m$, $m \geq 0$, such that $A \stackrel{\pi_1}{\Rightarrow} \cdots \stackrel{\pi_m}{\Rightarrow} w$ for some A and w.

We write $\alpha \Rightarrow^* \beta$ or $\alpha \Rightarrow^+ \beta$ to denote the existence of a string $\pi_1 \cdots \pi_m$ such that $\alpha \stackrel{\pi_1}{\Rightarrow} \cdots \stackrel{\pi_m}{\Rightarrow} \beta$, with $m \geq 0$ or $m > 0$, respectively.

For a CFG $\mathcal{G}$ we define the language $L(\mathcal{G})$ it generates as the set of strings w such that there is at least one derivation of w. We say a CFG is *reduced* if for each rule $\pi \in R$ there is a complete derivation in which it occurs.

A *probabilistic* context-free grammar (PCFG) is a pair $(\mathcal{G}, p)$ consisting of a CFG $\mathcal{G} = (\Sigma, N, S, R)$ and a probability function p from R to real numbers in the interval $[0, 1]$. We say a PCFG is *proper* if $\Sigma_{\pi=(A\rightarrow\gamma)\in R}\ p(\pi) = 1$ for each $A \in N$. PCFGs that arise from common methods of corpus linguistics, such as relative frequency estimation (Chi and Geman (1998)), are proper by construction.

For a PCFG $(\mathcal{G}, p)$, we define the probability $p(d)$ of a string $d = \pi_1 \cdots \pi_m \in R^*$ as $\prod_{i=1}^{m}\ p(\pi_i)$; we will in particular consider the probabilities of derivations d. The probability $p(w)$ of a string $w \in \Sigma^*$ as defined by $(\mathcal{G}, p)$ is the sum of the probabilities of all derivations of that string. We say a PCFG $(\mathcal{G}, p)$ is *consistent* if $\Sigma_{w\in\Sigma^*}\ p(w) = 1$.

In this paper we will mainly consider push-down transducers rather than push-down automata. Push-down transducers not only compute derivations of the grammar while processing an input string, but they also explicitly produce output strings from which these derivations can be obtained. We use transducers for two reasons. First, constraints on the output strings allow us to restrict our attention to 'reasonable' parsing strategies. Those strategies that cannot be formalized within these constraints are unlikely to be of practical interest. Secondly, mappings from input strings to derivations, such as those realized by push-down devices, turn out to be a very powerful abstraction and allow direct proofs of several general results.

Contrary to many textbooks, our push-down devices do not possess states next to stack symbols. This is without loss of generality, since states can be encoded into the stack symbols, given the types of transitions that we allow. Thus, a push-down transducer (PDT) $\mathcal{A}$ is a 6-tuple $(\Sigma_1, \Sigma_2, Q, X_{init}, X_{final}, \Delta)$, where Σ_1 is the input alphabet, Σ_2 is the output alphabet, Q is a finite set of *stack symbols* including the *initial stack symbol* X_{init} and the *final stack symbol* X_{final}, and Δ is the set of *transitions*. Each transition can have one of the following three forms: $X \mapsto XY$ (a push transition), $YX \mapsto Z$ (a pop transition), or $X \stackrel{x,y}{\mapsto} Y$ (a swap transition); here $X, Y, Z \in Q$, $x \in \Sigma_1 \cup \{\epsilon\}$ and $y \in \Sigma_2^*$. Note that in our notation, stacks grow from left to right, i.e., the top-most stack symbol will be found at the right end.

Without loss of generality, we assume that any PDT is such that for a given stack symbol $X \neq X_{final}$, there are either one or more push transitions $X \mapsto XY$, or one or more pop transitions $YX \mapsto Z$, or one or more swap transitions $X \stackrel{x,y}{\mapsto} Y$, but no combinations of different types of transitions. If a PDT does not satisfy this normal form, it can easily be brought in this form by introducing for each stack symbol X three new stack symbols X_{push}, X_{pop} and X_{swap} and new swap transitions $X \stackrel{\epsilon,\epsilon}{\mapsto} X_{push}$, $X \stackrel{\epsilon,\epsilon}{\mapsto} X_{pop}$ and $X \stackrel{\epsilon,\epsilon}{\mapsto} X_{swap}$. In each existing transition that operates on top-of-stack X, we then replace X by one from X_{push}, X_{pop} or X_{swap}, depending on the type of that transition. We also assume that X_{final} does not occur in the left-hand side of a transition, again without loss of generality.

A *configuration* of a PDT is a triple (α, w, v), where $\alpha \in Q^*$ is a stack, $w \in \Sigma_1^*$ is the remaining input, and $v \in \Sigma_2^*$ is the output generated so far. For a fixed PDT $\mathcal{A}$, we define the relation $\vdash$ on triples consisting of two configurations and a transition τ by: $(\gamma\alpha, xw, v) \overset{\tau}{\vdash} (\gamma\beta, w, vy)$ if and only if τ is of the form $\alpha \mapsto \beta$, where $x = y = \epsilon$, or of the form $\alpha \overset{x,y}{\mapsto} \beta$. A *computation* on an input string w is a string $c = \tau_1 \cdots \tau_m$, $m \geq 0$, such that $(X_{init}, w, \epsilon) \overset{\tau_1}{\vdash} \cdots \overset{\tau_m}{\vdash} (\alpha, w', v)$. A *complete* computation on a string w is a computation with $w' = \epsilon$ and $\alpha = X_{final}$. The string v is called the *output* of the computation c, and is denoted by $out(c)$.

We will identify a computation with the sequence of configurations that arise in that computation, where the first configuration is determined by the context. We also write $(\alpha, w, v) \vdash^* (\beta, w', v')$ or $(\alpha, w, v) \overset{c}{\vdash}{}^* (\beta, w', v')$, for $\alpha, \beta \in Q^*$, $w, w' \in \Sigma_1^*$ and $v, v' \in \Sigma_2^*$, to indicate that (β, w', v') can be obtained from (α, w, v) by applying a sequence c of zero or more transitions; we refer to such a sequence c as a *subcomputation*. The function *out* is extended to subcomputations in the natural way.

For a PDT $\mathcal{A}$, we define the language $L(\mathcal{A})$ it accepts as the set of strings w such that there is at least one complete computation on w. We say a PDT is *reduced* if each transition $\tau \in \Delta$ occurs in some complete computation.

A *probabilistic* push-down transducer (PPDT) is a pair $(\mathcal{A}, p)$ consisting of a PDT $\mathcal{A}$ and a probability function p from the set Δ of transitions of $\mathcal{A}$ to real numbers in the interval $[0, 1]$. We say a PPDT $(\mathcal{A}, p)$ is *proper* if

- $\Sigma_{\tau=(X \mapsto XY) \in \Delta}\, p(\tau) = 1$ for each $X \in Q$ such that there is at least one transition $X \mapsto XY$, $Y \in Q$;
- $\Sigma_{\tau=(X \overset{x,y}{\mapsto} Y) \in \Delta}\, p(\tau) = 1$ for each $X \in Q$ such that there is at least one transition $X \overset{x,y}{\mapsto} Y$, $x \in \Sigma_1 \cup \{\epsilon\}, y \in \Sigma_2^*, Y \in Q$; and
- $\Sigma_{\tau=(YX \mapsto Z) \in \Delta}\, p(\tau) = 1$, for each $X, Y \in Q$ such that there is at least one transition $YX \mapsto Z$, $Z \in Q$.

As in the case of PCFGs, we may expect a PPDT to be proper if its probability function p was obtained through a common method of corpus linguistics.

For a PPDT $(\mathcal{A}, p)$, we define the probability $p(c)$ of a (sub)computation $c = \tau_1 \cdots \tau_m$ as $\prod_{i=1}^{m} p(\tau_i)$. The probability $p(w)$ of a string w as defined by $(\mathcal{A}, p)$ is the sum of the probabilities of all complete computations on that string. We say a PPDT $(\mathcal{A}, p)$ is *consistent* if $\Sigma_{w \in \Sigma^*}\, p(w) = 1$.

We say a PCFG $(\mathcal{G}, p)$ is reduced if $\mathcal{G}$ is reduced, and we say a PPDT $(\mathcal{A}, p)$ is reduced if $\mathcal{A}$ is reduced.

3 Parsing Strategies

The term 'parsing strategy' is often used informally to refer to a class of parsing algorithms that behave similarly in some way. In this paper, we assign a formal meaning to this term, relying on the observation by Lang (1974) and Billot and Lang (1989) that many parsing algorithms for CFGs can be described in two steps. The first is a construction of push-down devices from CFGs, and the second is a method for handling nondeterminism (e.g. backtracking or dynamic programming). Parsing algorithms that handle nondeterminism in different ways but apply the same construction of push-down devices from CFGs are seen as realizations of the same parsing strategy.

Thus, we define a *parsing strategy* to be a function $\mathcal{S}$ that maps a reduced CFG $\mathcal{G} = (\Sigma_1, N, S, R)$ to a pair $\mathcal{S}(\mathcal{G}) = (\mathcal{A}, f)$ consisting of a reduced PDT $\mathcal{A} = (\Sigma_1, \Sigma_2, Q, X_{init}, X_{final}, \Delta)$, and a function f that maps a subset of Σ_2^* to a subset of R^*, with the following properties:

- $R \subseteq \Sigma_2$.

- For each string $w \in \Sigma_1^*$ and each complete computation c on w, $f(out(c)) = d$ is a derivation of w. Furthermore, each symbol from R occurs as often in $out(c)$ as it occurs in d.
- Conversely, for each string $w \in \Sigma_1^*$ and each derivation d of w, there is precisely one complete computation c on w such that $f(out(c)) = d$.

If c is a complete computation, we will write $f(c)$ to denote $f(out(c))$. The conditions above then imply that f is a bijection from complete computations to complete derivations.

Note that output strings of (complete) computations may contain symbols that are not in R, and the symbols that are in R may occur in a different order in v than in $f(v) = d$. The purpose of the symbols in $\Sigma_2 - R$ is to help this process of reordering of symbols in R. For a string $v \in \Sigma_2^*$ we let $\overline{v}$ refer to the maximal subsequence of symbols from v that belong to R, or in other words, string $\overline{v}$ is obtained by erasing from v all occurrences of symbols from $\Sigma_2 - R$.

A *probabilistic parsing strategy* is defined to be a function $\mathcal{S}$ that maps a reduced, proper and consistent PCFG $(\mathcal{G}, p_\mathcal{G})$ to a triple $\mathcal{S}(\mathcal{G}, p_\mathcal{G}) = (\mathcal{A}, p_\mathcal{A}, f)$, where $(\mathcal{A}, p_\mathcal{A})$ is a reduced, proper and consistent PPDT, with the same properties as a (non-probabilistic) parsing strategy, and in addition:

- For each complete derivation d and each complete computation c such that $f(c) = d$, $p_\mathcal{G}(d)$ equals $p_\mathcal{A}(c)$.

In other words, a complete computation has the same probability as the complete derivation that it is mapped to by function f. An implication of this property is that for each string $w \in \Sigma_1^*$, the probabilities assigned to that string by $(\mathcal{G}, p_\mathcal{G})$ and $(\mathcal{A}, p_\mathcal{A})$ are equal.

We say that probabilistic parsing strategy $\mathcal{S}'$ is an *extension* of parsing strategy $\mathcal{S}$ if for each reduced CFG $\mathcal{G}$ and probability function $p_\mathcal{G}$ we have $\mathcal{S}(\mathcal{G}) = (\mathcal{A}, f)$ if and only if $\mathcal{S}'(\mathcal{G}, p_\mathcal{G}) = (\mathcal{A}, p_\mathcal{A}, f)$ for some $p_\mathcal{A}$.

In the following sections we will investigate which parsing strategies can be extended to become probabilistic parsing strategies.

4 Correct-Prefix Property

For a given PDT, we say a computation c is *dead* if $(X_{init}, w_1, \epsilon) \stackrel{c}{\vdash^*} (\alpha, \epsilon, v_1)$, for some $\alpha \in Q^*$, $w_1 \in \Sigma_1^*$ and $v_1 \in \Sigma_2^*$, and there are no $w_2 \in \Sigma_1^*$ and $v_2 \in \Sigma_2^*$ such that $(\alpha, w_2, \epsilon) \vdash^* (X_{final}, \epsilon, v_2)$. Informally, a dead computation is a computation that cannot be continued to become a complete computation.

We say that a PDT has the *correct-prefix property* (CPP) if it does not allow any dead computations. We say that a parsing strategy has the CPP if it maps each reduced CFG to a PDT that has the CPP.

In this section we show that the correct-prefix property is a necessary condition for extending a parsing strategy to a probabilistic parsing strategy. For this we need two lemmas.

Lemma 1 *For each reduced CFG $\mathcal{G}$, there is a probability function $p_\mathcal{G}$ such that PCFG $(\mathcal{G}, p_\mathcal{G})$ is proper and consistent, and $p_\mathcal{G}(d) > 0$ for all complete derivations d.*

Proof. Since $\mathcal{G}$ is reduced, there is a finite set D consisting of complete derivations d, such that for each rule π in $\mathcal{G}$ there is at least one $d \in D$ in which π occurs. Let $n_{\pi,d}$ be the number of occurrences of rule π in derivation $d \in D$, and let n_π be $\Sigma_{d \in D}\, n_{\pi,d}$, the total number of occurrences of π in D. Let n_A be the sum of n_π for all rules π with A in the left-hand side. A probability function $p_\mathcal{G}$ can be defined through 'maximum-likelihood estimation' such that $p_\mathcal{G}(\pi) = \frac{n_\pi}{n_A}$ for each rule $\pi = A \rightarrow \alpha$.

For all nonterminals A, $\Sigma_{\pi = A \rightarrow \alpha}\, p_\mathcal{G}(\pi) = \Sigma_{\pi = A \rightarrow \alpha}\, \frac{n_\pi}{n_A} = \frac{n_A}{n_A} = 1$, which means that the PCFG $(\mathcal{G}, p_\mathcal{G})$ is proper. Furthermore, Chi and Geman (1998) has shown that a PCFG $(\mathcal{G}, p_\mathcal{G})$ is consistent if $p_\mathcal{G}$ was obtained by maximum-likelihood estimation using a set of derivations. Finally, since $n_\pi > 0$ for each π, also $p_\mathcal{G}(\pi) > 0$ for each π, and $p_\mathcal{G}(d) > 0$ for all complete derivations d. ∎

We say a computation is a *shortest* dead computation if it is dead and none of its proper prefixes is dead. Note that each dead computation has a unique prefix that is a shortest dead computation. For a PDT $\mathcal{A}$, let $\mathcal{T}_\mathcal{A}$ be the union of the set of all complete computations and the set of all shortest dead computations.

Lemma 2 *For each proper PPDT* $(\mathcal{A}, p_\mathcal{A})$, $\Sigma_{c \in \mathcal{T}_\mathcal{A}}\ p_\mathcal{A}(c) \leq 1$.

Proof. The proof is a trivial variant of the proof that for a proper PCFG $(\mathcal{G}, p_\mathcal{G})$, the sum of $p_\mathcal{G}(d)$ for all derivations d cannot exceed 1, which is shown by Booth and Thompson (1973). ■

From this, the main result of this section follows.

Theorem 3 *A parsing strategy that lacks the CPP cannot be extended to become a probabilistic parsing strategy.*

Proof. Take a parsing strategy $\mathcal{S}$ that does not have the CPP. Then there is a reduced CFG $\mathcal{G} = (\Sigma_1, N, S, R)$, with $\mathcal{S}(\mathcal{G}) = (\mathcal{A}, f)$ for some $\mathcal{A}$ and f, and a shortest dead computation c allowed by $\mathcal{A}$.

It follows from Lemma 1 that there is a probability function $p_\mathcal{G}$ such that $(\mathcal{G}, p_\mathcal{G})$ is a proper and consistent PCFG and $p_\mathcal{G}(d) > 0$ for all complete derivations d. Assume we also have a probability function $p_\mathcal{A}$ such that $(\mathcal{A}, p_\mathcal{A})$ is a proper and consistent PPDT and $p_\mathcal{A}(c') = p_\mathcal{G}(f(c'))$ for each complete computation c'. Since $\mathcal{A}$ is reduced, each transition τ must occur in some complete computation c'. Furthermore, for each complete computation c' there is a complete derivation d such that $f(c') = d$, and $p_\mathcal{A}(c') = p_\mathcal{G}(d) > 0$. Therefore, $p_\mathcal{A}(\tau) > 0$ for each transition τ, and $p_\mathcal{A}(c) > 0$, where c is the above-mentioned dead computation.

Due to Lemma 2, $1 \geq \Sigma_{c' \in \mathcal{T}_\mathcal{A}}\ p_\mathcal{A}(c') \geq \Sigma_{w \in \Sigma_1^*}\ p_\mathcal{A}(w) + p_\mathcal{A}(c) > \Sigma_{w \in \Sigma_1^*}\ p_\mathcal{A}(w) = \Sigma_{w \in \Sigma_1^*}\ p_\mathcal{G}(w)$. This is in contradiction with the consistency of $(\mathcal{G}, p_\mathcal{G})$. Hence, a probability function $p_\mathcal{A}$ with the properties we required above cannot exist, and therefore $\mathcal{S}$ cannot be extended to become a probabilistic parsing strategy. ■

5 Strong Predictiveness

For a fixed PDT, we define the binary relation $\rightsquigarrow$ on stack symbols by: $Y \rightsquigarrow Y'$ if and only if $(Y, w, \epsilon) \vdash^* (Y', \epsilon, v)$ for some $w \in \Sigma_1^*$ and $v \in \Sigma_2^*$. In other words, some subcomputation may start with stack Y and end with stack Y'. Note that all stacks that occur in such a subcomputation must have height of 1 or more.

We say that a PDT has the *strong predictiveness property* (SPP) if the existence of three transitions $X \mapsto XY$, $XY_1 \mapsto Z_1$ and $XY_2 \mapsto Z_2$ such that $Y \rightsquigarrow Y_1$ and $Y \rightsquigarrow Y_2$ implies $Z_1 = Z_2$. Informally, this means that when a subcomputation starts with some stack α and some push transition τ, then solely on the basis of τ we can uniquely determine what stack symbol $Z_1 = Z_2$ will be on top of the stack in the first configuration with stack height equal to $|\alpha|$. Another way of looking at it is that no information may flow from higher stack elements to lower stack elements that was not already predicted before these higher stack elements came into being, hence the term 'strong predictiveness'.[2]

We say that a parsing strategy has the SPP if it maps each reduced CFG to a PDT with the SPP.

In the previous section it was shown that we may restrict ourselves to parsing strategies that have the CPP. Here we show that if, in addition, a parsing strategy has the SPP, then it can always be extended to become a probabilistic parsing strategy.

Theorem 4 *Any parsing strategy that has the CPP and the SPP can be extended to become a probabilistic parsing strategy.*

[2]There is a property of push-down devices called *faiblement prédictif* (weakly predictive) (Villemonte de la Clergerie (1993)). Contrary to what this name may suggest however, this property is incomparable with the complement of our notion of SPP.

Proof. Take a parsing strategy $\mathcal{S}$ that has the CPP and the SPP, and take a reduced PCFG $(\mathcal{G}, p_{\mathcal{G}})$, where $\mathcal{G} = (\Sigma, N, S, R)$, and let $\mathcal{S}(\mathcal{G}) = (\mathcal{A}, f)$, for some PDT $\mathcal{A}$ and function f. We will show that there is a probability function $p_{\mathcal{A}}$ such that $(\mathcal{A}, p_{\mathcal{A}})$ is a PPDT and $p_{\mathcal{A}}(c) = p_{\mathcal{G}}(f(c))$ for all complete computations c.

For each stack symbol X, consider the set of transitions that are applicable with top-of-stack X. Remember that our normal form ensures that all such transitions are of the same type. Suppose this set consists of m swap transitions $\tau_i = X \overset{x_i, y_i}{\mapsto} Y_i$, $1 \leq i \leq m$. For each i, consider all subcomputations of the form $(X, x_i w, \epsilon) \overset{\tau_i}{\vdash} (Y_i, w, y_i) \vdash^* (Y', \epsilon, v)$ such that there is at least one pop transition of the form $ZY' \mapsto Z'$ or such that $Y' = X_{\mathit{final}}$, and define L_{τ_i} as the set of strings v output by these subcomputations. We also define $L_X = \cup_{j=1}^{m} L_{\tau_j}$, the set of all strings output by subcomputations starting with top-of-stack X, and ending just before a pop transition that leads to a stack with height smaller than that of the stack at the beginning, or ending with the final stack symbol X_{final}.

Now define for each i ($1 \leq i \leq m$):

$$p_{\mathcal{A}}(\tau_i) \quad = \quad \frac{\Sigma_{v \in L_{\tau_i}} \; p_{\mathcal{G}}(\overline{v})}{\Sigma_{v \in L_X} \; p_{\mathcal{G}}(\overline{v})} \tag{1}$$

In other words, the probability of a transition is the normalized probability of the set of subcomputations starting with that transition, relating subcomputations with fragments of derivations of the PCFG.

These definitions are well-defined. Since $\mathcal{A}$ is reduced and has the CPP, the sets L_{τ_i} are non-empty and thereby the denominator in the definition of $p_{\mathcal{A}}(\tau_i)$ is non-zero. Furthermore, $\Sigma_{i=1}^{m} p_{\mathcal{A}}(\tau_i)$ is clearly 1.

Now suppose the set of transitions for X consists of m push transitions $\tau_i = X \mapsto XY_i$, $1 \leq i \leq m$. For each i, consider all subcomputations of the form $(X, w, \epsilon) \overset{\tau_i}{\vdash} (XY_i, w, \epsilon) \vdash^* (X', \epsilon, v)$ such that there is at least one pop transition of the form $ZX' \mapsto Z'$ or $X' = X_{\mathit{final}}$, and define L_{τ_i}, L_X and $p_{\mathcal{A}}(\tau_i)$ as we have done above for the swap transitions.

Suppose the set of transitions for X consists of m pop transitions $\tau_i = Y_i X \mapsto Z_i$, $1 \leq i \leq m$. Define $L_X = \{\epsilon\}$, and $p_{\mathcal{A}}(\tau_i) = 1$ for each i. To see that this is compatible with the condition of properness of PPDTs, note the following. Since we may assume $\mathcal{A}$ is reduced, if $Y_i = Y_j$ for some i and j with $1 \leq i, j \leq m$, then there is at least one transition $Y_i \mapsto Y_i X'$ for some X' such that $X' \rightsquigarrow X$. Due to the SPP, $Z_i = Z_j$ and therefore $i = j$.

Finally, we define $L_{X_{\mathit{final}}} = \{\epsilon\}$.

Take a subcomputation $(X, w, \epsilon) \overset{c}{\vdash^*} (Y, \epsilon, v)$ such that there is at least one pop transition of the form $ZY \mapsto Y'$ or $Y = X_{\mathit{final}}$. Below we will prove that:

$$p_{\mathcal{A}}(c) \quad = \quad \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_X} \; p_{\mathcal{G}}(\overline{v'})} \tag{2}$$

Since a complete computation c with output v is of this form, with $X = X_{\mathit{init}}$ and $Y = X_{\mathit{final}}$, we obtain the result we required to prove Theorem 4, where D denotes the set of all complete derivations of CFG $\mathcal{G}$:

$$\begin{aligned} p_{\mathcal{A}}(c) &= \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_{X_{\mathit{init}}}} \; p_{\mathcal{G}}(\overline{v'})} \\ &= \frac{p_{\mathcal{G}}(f(c))}{\Sigma_{v' \in L_{X_{\mathit{init}}}} \; p_{\mathcal{G}}(f(v'))} \\ &= \frac{p_{\mathcal{G}}(f(c))}{\Sigma_{d \in D} \; p_{\mathcal{G}}(d)} \\ &= p_{\mathcal{G}}(f(c)) \end{aligned}$$

We have used two properties of f here. The first is that it preserves the frequencies of symbols from R, if considered as a mapping from output strings to derivations. The second property is

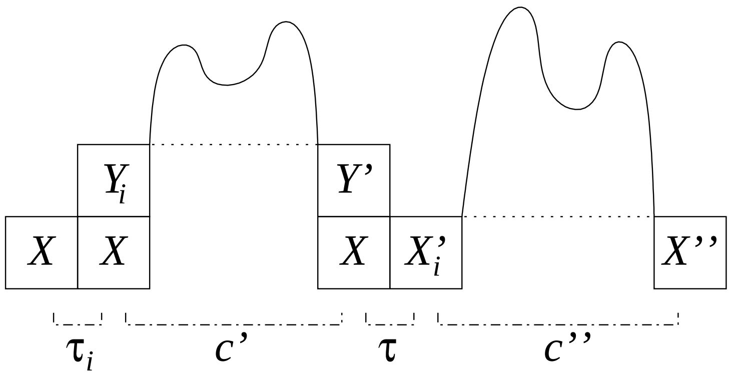

Figure 1: Development of the stack in the computation $c = \tau_i c' \tau c''$.

that it can be considered as bijection from complete computations to complete derivations. We also used consistency of PCFG $(\mathcal{G}, p_{\mathcal{G}})$, meaning that $\Sigma_{d \in D}\ p_{\mathcal{G}}(d) = 1$.

For the proof of (2), we proceed by induction on the length of c and distinguish three cases.

Case 1: Consider a subcomputation c consisting of zero transitions, which naturally has output $v = \epsilon$, with only configuration (X, ϵ, ϵ), where there is at least one pop transition of the form $ZX \mapsto Z'$ or $X = X_{\mathit{final}}$. We trivially have $p_{\mathcal{A}}(c) = 1$ and $\frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_X}\ p_{\mathcal{G}}(\overline{v'})} = \frac{p_{\mathcal{G}}(\epsilon)}{\Sigma_{v' \in \{\epsilon\}}\ p_{\mathcal{G}}(\overline{v'})} = 1$.

Case 2: Consider a subcomputation $c = \tau_i c'$, where $(X, x_i w, \epsilon) \overset{\tau_i}{\vdash} (Y_i, w, y_i) \overset{c'}{\vdash^*} (Y', \epsilon, y_i v)$, such that there is at least one pop transition of the form $ZY' \mapsto Z'$ or $Y' = X_{\mathit{final}}$. The induction hypothesis states that:

$$p_{\mathcal{A}}(c') \quad = \quad \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_{Y_i}}\ p_{\mathcal{G}}(\overline{v'})}$$

If we combine this with the definition of $p_{\mathcal{A}}$, we obtain:

$$\begin{aligned}
p_{\mathcal{A}}(c) &= p_{\mathcal{A}}(\tau_i) \cdot p_{\mathcal{A}}(c') \\
&= \frac{\Sigma_{v' \in L_{\tau_i}}\ p_{\mathcal{G}}(\overline{v'})}{\Sigma_{v' \in L_X}\ p_{\mathcal{G}}(\overline{v'})} \cdot \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_{Y_i}}\ p_{\mathcal{G}}(\overline{v'})} \\
&= \frac{p_{\mathcal{G}}(\overline{y_i}) \cdot \Sigma_{v' \in L_{Y_i}}\ p_{\mathcal{G}}(\overline{v'})}{\Sigma_{v' \in L_X}\ p_{\mathcal{G}}(\overline{v'})} \cdot \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_{Y_i}}\ p_{\mathcal{G}}(\overline{v'})} \\
&= \frac{p_{\mathcal{G}}(\overline{y_i}) \cdot p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_X}\ p_{\mathcal{G}}(\overline{v'})} \\
&= \frac{p_{\mathcal{G}}(\overline{y_i v})}{\Sigma_{v' \in L_X}\ p_{\mathcal{G}}(\overline{v'})}
\end{aligned}$$

Case 3: Consider a subcomputation c of the form $(X, w, \epsilon) \overset{\tau_i}{\vdash} (XY_i, w, \epsilon) \vdash^* (X'', \epsilon, v)$ such that there is at least one pop transition of the form $ZX'' \mapsto Z'$ or $X'' = X_{\mathit{final}}$. Subcomputation c can be decomposed in a unique way as $c = \tau_i c' \tau c''$, consisting of an application of a push transition $\tau_i = X \mapsto XY_i$, a subcomputation $(Y_i, w_1, \epsilon) \overset{c'}{\vdash^*} (Y', \epsilon, v_1)$, an application of a pop transition $\tau = XY' \mapsto X'_i$, and a subcomputation $(X'_i, w_2, \epsilon) \overset{c''}{\vdash^*} (X'', \epsilon, v_2)$, where $w = w_1 w_2$ and $v = v_1 v_2$. This is visualized in Figure 1.

We can now use the induction hypothesis twice, resulting in:

$$p_{\mathcal{A}}(c') \quad = \quad \frac{p_{\mathcal{G}}(\overline{v_1})}{\Sigma_{v'_1 \in L_{Y_i}}\ p_{\mathcal{G}}(\overline{v'_1})}$$

and

$$p_{\mathcal{A}}(c'') = \frac{p_{\mathcal{G}}(\overline{v_2})}{\Sigma_{v_2' \in L_{X_i'}} \; p_{\mathcal{G}}(\overline{v_2'})}$$

If we combine this with the definition of $p_{\mathcal{A}}$, we obtain:

$$\begin{aligned} p_{\mathcal{A}}(c) &= p_{\mathcal{A}}(\tau_i) \cdot p_{\mathcal{A}}(c') \cdot p_{\mathcal{A}}(\tau) \cdot p_{\mathcal{A}}(c'') \\ &= \frac{\Sigma_{v' \in L_{\tau_i}} \; p_{\mathcal{G}}(\overline{v'})}{\Sigma_{v' \in L_X} \; p_{\mathcal{G}}(\overline{v'})} \cdot \frac{p_{\mathcal{G}}(\overline{v_1})}{\Sigma_{v_1' \in L_{Y_i}} \; p_{\mathcal{G}}(\overline{v_1'})} \cdot \\ & \qquad 1 \cdot \frac{p_{\mathcal{G}}(\overline{v_2})}{\Sigma_{v_2' \in L_{X_i'}} \; p_{\mathcal{G}}(\overline{v_2'})} \end{aligned}$$

Since $\mathcal{A}$ has the SPP, X_i' is unique to τ_i and the output strings in L_{τ_i} are precisely those that can be obtained by concatenating an output string in L_{Y_i} and an output string in $L_{X_i'}$. Therefore $\Sigma_{v' \in L_{\tau_i}} \; p_{\mathcal{G}}(\overline{v'}) = \Sigma_{v_1' \in L_{Y_i}} \Sigma_{v_2' \in L_{X_i'}} \; p_{\mathcal{G}}(\overline{v_1' v_2'}) = \Sigma_{v_1' \in L_{Y_i}} \; p_{\mathcal{G}}(\overline{v_1'}) \cdot \Sigma_{v_2' \in L_{X_i'}} \; p_{\mathcal{G}}(\overline{v_2'})$, and

$$\begin{aligned} p_{\mathcal{A}}(c) &= \frac{p_{\mathcal{G}}(\overline{v_1}) \cdot p_{\mathcal{G}}(\overline{v_2})}{\Sigma_{v' \in L_X} \; p_{\mathcal{G}}(\overline{v'})} \\ &= \frac{p_{\mathcal{G}}(\overline{v_1 v_2})}{\Sigma_{v' \in L_X} \; p_{\mathcal{G}}(\overline{v'})} \\ &= \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_X} \; p_{\mathcal{G}}(\overline{v'})} \end{aligned}$$

This concludes the proof. ■

Note that the definition of $p_{\mathcal{A}}$ in the above proof relies on the strings output by $\mathcal{A}$. This is the main reason why we needed to consider push-down transducers rather than push-down automata (defined below). Now assume an appropriate probability function $p_{\mathcal{A}}$ has been found such that $(\mathcal{A}, p_{\mathcal{A}})$ is a PPDT that assigns the same probabilities to computations as the given PCFG assigns to the corresponding derivations, following the construction from the proof above. Then the probabilities assigned to strings over the input alphabet are also equal. We may subsequently ignore the output strings if the application at hand merely requires probabilistic recognition rather than probabilistic transduction, or in other words, we may simplify push-down transducers to push-down automata.

Formally, a *push-down automaton* (PDA) $\mathcal{A}$ is a 5-tuple $(\Sigma, Q, X_{init}, X_{final}, \Delta)$, where Σ is the input alphabet, and Q, X_{init}, X_{final} and Δ are as in the definition of PDTs. Push and pop transitions are as before, but swap transitions are simplified to the form $X \overset{x}{\mapsto} Y$, where $x \in \{\epsilon\} \cup \Sigma$. Computations are defined as in the case of PDTs, except that configurations are now pairs (α, w) whereas they were triples (α, w, v) in the case of PDTs. A *probabilistic* push-down automaton (PPDA) is a pair $(\mathcal{A}, p_{\mathcal{A}})$, where $\mathcal{A}$ is a PDA and $p_{\mathcal{A}}$ is a probability function subject to the same constraints as in the case of PPDTs. Since the definitions of CPP and SPP for PDTs did not refer to output strings, these notions carry over to PDAs in a straightforward way.

We define the size of a CFG as $\sum_{(A \rightarrow \alpha) \in R} |A\alpha|$, the total number of occurrences of terminals and nonterminals in the set of rules. Similarly, we define the size of a PDA as $\sum_{(\alpha \mapsto \beta) \in \Delta} |\alpha\beta| + \sum_{(X \overset{x}{\mapsto} Y) \in \Delta} |XxY|$, the total number of occurrences of stack symbols and terminals in the set of transitions.

Let $\mathcal{A} = (\Sigma, Q, X_{init}, X_{final}, \Delta)$ be a PDA with both CPP and SPP. We will now show that we can construct an equivalent CFG $\mathcal{G} = (\Sigma, Q, X_{init}, R)$ with size linear in the size of $\mathcal{A}$. The rules of this grammar are the following.

- $X \rightarrow YZ$ for each transition $X \mapsto XY$, where Z is the unique stack symbol such that there is at least one transition $XY' \mapsto Z$ with $Y \rightsquigarrow Y'$;

- $X \to xY$ for each transition $X \overset{x}{\mapsto} Y$;
- $Y \to \epsilon$ for each stack symbol Y such that there is at least one transition $XY \mapsto Z$ or such that $Y = X_{\mathit{final}}$.

It is easy to see that there exists a bijection from complete computations of $\mathcal{A}$ to complete derivations of $\mathcal{G}$, preserving the recognized/derived strings. Apart from an additional derivation step by rule $X_{\mathit{final}} \to \epsilon$, the complete derivations also have the same length as the corresponding complete computations.

The above construction can straightforwardly be extended to probabilistic PDAs (PPDAs). Let $(\mathcal{A}, p_{\mathcal{A}})$ be a PPDA with both CPP and SPP. Then we construct $\mathcal{G}$ as above, and further define $p_{\mathcal{G}}$ such that $p_{\mathcal{G}}(\pi) = p_{\mathcal{A}}(\tau)$ for rules $\pi = X \to YZ$ or $\pi = X \to xY$ that we construct out of transitions $\tau = X \mapsto XY$ or $\tau = X \overset{x}{\mapsto} Y$, respectively, in the first two items above. We also define $p_{\mathcal{G}}(Y \to \epsilon) = 1$ for rules $Y \to \epsilon$ obtained in the third item above. If $(\mathcal{A}, p_{\mathcal{A}})$ is reduced, proper and consistent then so is $(\mathcal{G}, p_{\mathcal{G}})$.

This leads to the observation that parsing strategies with the CPP and the SPP as well as their probabilistic extensions can also be described as grammar transformations, as follows. A given (P)CFG is mapped to an equivalent (P)PDT by a (probabilistic) parsing strategy. By ignoring the output components of swap transitions we obtain a (P)PDA, which can be mapped to an equivalent (P)CFG as shown above. This observation gives rise to an extension with probabilities of the work on *covers* by Nijholt (1980); Leermakers (1989).

The simple relationship between PDAs with both CPP and SPP on the one hand and CFGs on the other can be used to carry over algorithms originally designed for CFGs to PDAs or PDTs. One such application is the evaluation of the right-hand side of equation (1) in the proof of Theorem 4. Both the numerator and the denominator involve potentially infinite sets of subcomputations, and therefore it is not immediately clear that the proof is constructive. However, there are published algorithms to compute, for a given PCFG $(\mathcal{G}', p_{\mathcal{G}'})$ that is not necessarily proper and a given nonterminal A, the expression $\Sigma_{w \in \Sigma^*} p_{\mathcal{G}'}(A \Rightarrow^* w)$, or rather, to approximate it with arbitrary precision; see Booth and Thompson (1973) and Stolcke (1995). This can be used to compute e.g. $\Sigma_{v \in L_X} p_{\mathcal{G}}(\overline{v})$ in equation (1), as follows.

The first step is to map the PDT to a CFG $\mathcal{G}'$ as shown above. We then define a function $p_{\mathcal{G}'}$ that assigns probability 1 to all rules that we construct out of push and pop transitions. We also let $p_{\mathcal{G}'}$ assign probability $p_{\mathcal{G}}(\overline{y})$ to a rule $X \to xY$ that we construct out of a scan transition $X \overset{x,y}{\mapsto} Y$. It is easy to see that, for any stack symbol X, we have $\Sigma_{v \in L_X} p_{\mathcal{G}}(\overline{v}) = \Sigma_{w \in \Sigma_1^*} p_{\mathcal{G}'}(X \Rightarrow^* w)$. This allows our problem on the computations of probabilities in the right-hand side of equation (1) to be reduced to a problem on PCFGs, which can be solved by existing algorithms as discussed above.

The same idea can be used to show that determination of $p_{\mathcal{A}}$ by equation (1) can be seen as an application of PCFG renormalization (Abney et al. (1999); Chi (1999); Nederhof and Satta (2003)), which allows an alternative proof of Theorem 4. Our proof seems more insightful however, especially with regard to the reason why some parsing strategies *without* the SPP cannot be extended to become probabilistic, as we will show in the next section.

6 Examples of Parsing Strategies

Many well-known parsing strategies with the CPP also have the SPP, such as top-down parsing (Harrison (1978)) and left-corner parsing (Rosenkrantz and Lewis II (1970)). It was already shown by Tendeau (1995) that for any PCFG $\mathcal{G}$, there is an equivalent probabilistic push-down automaton that implements the left-corner strategy, of which the size is $\mathcal{O}(|\mathcal{G}|^2)$, where $|\mathcal{G}|$ denotes the size of $\mathcal{G}$.[3] Also PLR parsing (Soisalon-Soininen and Ukkonen (1979)) can be expressed in our framework as a parsing strategy with the CPP and the SPP.

[3] This is in contrast to a construction of 'shift-reduce' probabilistic push-down automata out of PCFGs from Abney et al. (1999), which were of size $\mathcal{O}(|\mathcal{G}|^5)$. The greater size is mainly due to the restrictive type of automaton considered there, which does not allow for all types of parsing strategies.

These strategies are in contrast to LR parsing, which is a parsing strategy with the CPP but without the SPP, and in fact, LR parsing cannot be extended to become a probabilistic parsing strategy, as we will show. Below, we assume the reader is familiar with LR parsing (Sippu and Soisalon-Soininen (1990)).

We take a PCFG $(\mathcal{G}, p_{\mathcal{G}})$ defined by:

$$\begin{array}{lll} \pi_S & = S \to AB, & p_{\mathcal{G}}(\pi_S) = 1 \\ \pi_{A_1} & = A \to aC, & p_{\mathcal{G}}(\pi_{A_1}) = \frac{1}{3} \\ \pi_{A_2} & = A \to aD, & p_{\mathcal{G}}(\pi_{A_2}) = \frac{2}{3} \\ \pi_{B_1} & = B \to bC, & p_{\mathcal{G}}(\pi_{B_1}) = \frac{2}{3} \\ \pi_{B_2} & = B \to bD, & p_{\mathcal{G}}(\pi_{B_2}) = \frac{1}{3} \\ \pi_C & = C \to xc, & p_{\mathcal{G}}(\pi_C) = 1 \\ \pi_D & = D \to xd, & p_{\mathcal{G}}(\pi_D) = 1 \end{array}$$

Note that this grammar generates a finite language.

We will not present the entire LR automaton $\mathcal{A}$, with $\mathcal{S}_{LR}(\mathcal{G}) = (\mathcal{A}, f)$ for some f, but we merely mention two of its key transitions, which represent shift actions over c and d:

$$\begin{array}{ll} \tau_c = & \{C \to x \bullet c, D \to x \bullet d\} \overset{c,\epsilon}{\mapsto} \\ & \{C \to x \bullet c, D \to x \bullet d\}\ \{C \to xc\, \bullet\} \\ \tau_d = & \{C \to x \bullet c, D \to x \bullet d\} \overset{d,\epsilon}{\mapsto} \\ & \{C \to x \bullet c, D \to x \bullet d\}\ \{D \to xd\, \bullet\} \end{array}$$

(We denote LR states by their sets of kernel items, as usual.)

Take a probability function $p_{\mathcal{A}}$ such that $(\mathcal{A}, p_{\mathcal{A}})$ is a proper PPDT. It can be easily seen that $p_{\mathcal{A}}$ must assign 1 to all transitions except τ_c and τ_d, since that is the only pair of distinct transitions that can be applied for one and the same top-of-stack symbol, viz. $\{C \to x \bullet c, D \to x \bullet d\}$.

However, $\frac{p_{\mathcal{G}}(axcbxd)}{p_{\mathcal{G}}(axdbxc)} = \frac{p_{\mathcal{G}}(\pi_{A_1}) \cdot p_{\mathcal{G}}(\pi_{B_2})}{p_{\mathcal{G}}(\pi_{A_2}) \cdot p_{\mathcal{G}}(\pi_{B_1})} = \frac{\frac{1}{3} \cdot \frac{1}{3}}{\frac{2}{3} \cdot \frac{2}{3}} = \frac{1}{4}$ but $\frac{p_{\mathcal{A}}(axcbxd)}{p_{\mathcal{A}}(axdbxc)} = \frac{p_{\mathcal{A}}(\tau_c) \cdot p_{\mathcal{A}}(\tau_d)}{p_{\mathcal{A}}(\tau_d) \cdot p_{\mathcal{A}}(\tau_c)} = 1 \neq \frac{1}{4}$. This shows that there is no $p_{\mathcal{A}}$ such that $(\mathcal{A}, p_{\mathcal{A}})$ assigns the same probabilities to strings over Σ as $(\mathcal{G}, p_{\mathcal{G}})$. It follows that the LR strategy cannot be extended to become a probabilistic parsing strategy.

Note that for $\mathcal{G}$ as above, $p_{\mathcal{G}}(\pi_{A_1})$ and $p_{\mathcal{G}}(\pi_{B_1})$ can be freely chosen, and this choice determines the other values of $p_{\mathcal{G}}$, so we have two free parameters. For $\mathcal{A}$ however, there is only one free parameter in the choice of $p_{\mathcal{A}}$. This is in conflict with an underlying assumption of existing work on probabilistic LR parsing, by e.g. Briscoe and Carroll (1993) and Inui et al. (2000), viz. that LR parsers would allow more fine-grained probability distributions than CFGs. However, for some practical grammars from the area of natural language processing, Sornlertlamvanich et al. (1999) have shown that LR parsers do allow more accurate probability distributions than the CFGs from which they were constructed, if probability functions are estimated from corpora.

Related to LR parsing is ELR parsing (Purdom and Brown (1981); Nederhof (1994)), which also lacks the SPP. By an argument similar to the one above, we can show that also ELR cannot be extended to become a probabilistic parsing strategy. That ELR parsing could not be extended to become a probabilistic parsing strategy in any straightforward way has already been remarked by Tendeau (1997).

7 Conclusions

We have formalized the notion of parsing strategy as a mapping from context-free grammars to push-down transducers, and have investigated the extension to probabilities. We have shown that the question of which parsing strategies can be extended to become probabilistic heavily relies on two properties, the correct-prefix property and the strong predictiveness property. The CPP is a necessary condition for extending a strategy to become a probabilistic strategy. The CPP and SPP together form a sufficient condition. We have shown that there is at least one strategy of practical interest with the CPP but without the SPP that cannot be extended to become a probabilistic strategy.

References

Abney, S., McAllester, D., and Pereira, F. (1999). Relating probabilistic grammars and automata. In *37th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 542–549, Maryland, USA.

Billot, S. and Lang, B. (1989). The structure of shared forests in ambiguous parsing. In *27th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 143–151, Vancouver, British Columbia, Canada.

Booth, T. and Thompson, R. (1973). Applying probabilistic measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450.

Briscoe, T. and Carroll, J. (1993). Generalized probabilistic LR parsing of natural language (corpora) with unification-based grammars. *Computational Linguistics*, 19(1):25–59.

Charniak, E. and Carroll, G. (1994). Context-sensitive statistics for improved grammatical language models. In *Proceedings Twelfth National Conference on Artificial Intelligence*, volume 1, pages 728–733, Seattle, Washington.

Chi, Z. (1999). Statistical properties of probabilistic context-free grammars. *Computational Linguistics*, 25(1):131–160.

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

Chitrao, M. and Grishman, R. (1990). Statistical parsing of messages. In *Speech and Natural Language, Proceedings*, pages 263–266, Hidden Valley, Pennsylvania.

Harrison, M. (1978). *Introduction to Formal Language Theory.* Addison-Wesley.

Inui, K. et al. (2000). Probabilistic GLR parsing. In Bunt, H. and Nijholt, A., editors, *Advances in Probabilistic and other Parsing Technologies*, chapter 5, pages 85–104. Kluwer Academic Publishers.

Lang, B. (1974). Deterministic techniques for efficient non-deterministic parsers. In *Automata, Languages and Programming, 2nd Colloquium*, volume 14 of *Lecture Notes in Computer Science*, pages 255–269, Saarbrücken. Springer-Verlag.

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

Manning, C. and Schütze, H. (1999). *Foundations of Statistical Natural Language Processing.* Massachusetts Institute of Technology.

Nederhof, M.-J. (1994). An optimal tabular parsing algorithm. In *32nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 117–124, Las Cruces, New Mexico, USA.

Nederhof, M.-J. and Satta, G. (2003). Probabilistic parsing as intersection. In *8th International Workshop on Parsing Technologies*, pages 137–148, LORIA, Nancy, France.

Nijholt, A. (1980). *Context-Free Grammars: Covers, Normal Forms, and Parsing*, volume 93 of *Lecture Notes in Computer Science.* Springer-Verlag.

Purdom, Jr., P. and Brown, C. (1981). Parsing extended LR(k) grammars. *Acta Informatica*, 15:115–127.

Roark, B. and Johnson, M. (1999). Efficient probabilistic top-down and left-corner parsing. In *37th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 421–428, Maryland, USA.

Rosenkrantz, D. and Lewis II, P. (1970). Deterministic left corner parsing. In *IEEE Conference Record of the 11th Annual Symposium on Switching and Automata Theory*, pages 139–152.

Santos, E. (1972). Probabilistic grammars and automata. *Information and Control*, 21:27–47.

Sippu, S. and Soisalon-Soininen, E. (1990). *Parsing Theory, Vol. II: LR(k) and LL(k) Parsing*, volume 20 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag.

Soisalon-Soininen, E. and Ukkonen, E. (1979). A method for transforming grammars into LL(k) form. *Acta Informatica*, 12:339–369.

Sornlertlamvanich, V. et al. (1999). Empirical support for new probabilistic generalized LR parsing. *Journal of Natural Language Processing*, 6(3):3–22.

Stolcke, A. (1995). An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational Linguistics*, 21(2):167–201.

Tendeau, F. (1995). Stochastic parse-tree recognition by a pushdown automaton. In *Fourth International Workshop on Parsing Technologies*, pages 234–249, Prague and Karlovy Vary, Czech Republic.

Tendeau, F. (1997). *Analyse syntaxique et sémantique avec évaluation d'attributs dans un demi-anneau*. PhD thesis, University of Orléans.

Villemonte de la Clergerie, E. (1993). *Automates à Piles et Programmation Dynamique — DyALog: Une application à la Programmation en Logique*. PhD thesis, Université Paris VII.