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 so-called generalized LR parsing.

1 Introduction

Context-free grammars and push-down automata are two equivalent formalisms to describe context-free languages. While a context-free grammar can be thought of as a purely declarative specification, a push-down automaton 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 context-free grammars to equivalent push-down automata, such that some specific conditions are observed.

This paper deals with the probabilistic extensions of context-free grammars and push-down automata, i.e., probabilistic context-free grammars [41, 4] and probabilistic push-down automata [41, 42, 50, 1]. These formalisms are obtained

[*] Supported by the Royal Netherlands Academy of Arts and Sciences. Secondary affiliation is the German Research Center for Artificial Intelligence (DFKI).

by adding probabilities to the rules and transitions of context-free grammars and push-down automata, respectively. More specifically, we will investigate the problem of 'extending' parsing strategies to *probabilistic* parsing strategies. These are mappings from probabilistic context-free grammars to probabilistic push-down automata that preserve the induced probability distributions on the generated/accepted languages. Two of the main results presented in this paper can be stated as follows:

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

The above results generalize previous findings reported in [50, 51, 1], where only a few specific parsing strategies were considered in isolation. Our findings also have important implications for well-known parsing strategies such as generalized 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 the last decade, widespread interest in probabilistic parsing techniques has arisen in the area of natural language processing [6, 27, 21]. This is motivated by the fact that natural language sentences are generally ambiguous, and natural language software needs to be able to distinguish the more probable derivations of a sentence from the less probable ones. This can be achieved by letting the parsing process assign a probability to each parse, on the basis of a probabilistic grammar. In a typical application, the software may select those derivations for further processing that have been given the highest probabilities, and discard the others. The success of this approach relies on the accuracy of the probabilistic model expressed by the probabilistic grammar, i.e., whether the probabilities assigned to derivations accurately reflect the 'true' probabilities in the domain at hand.

Probabilities are often estimated on the basis of a corpus, i.e., a collection of sentences. The sentences in a corpus may be annotated with various kinds of information. One kind of annotation that is relevant for our discussion is the preferred derivation for each sentence. Given a corpus with derivations, one may estimate probabilities of rules by their relative frequencies in the corpus. If a corpus is unannotated, more general techniques of maximum-likelihood estimation can be used to estimate the probabilities of rules. (See [40, 11, 10] for some formal properties of types of maximum-likelihood estimation.)

The motivation for studying probabilistic models other than those obtained by attaching probabilities to given context-free grammars is the observation that

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

more accurate models can be obtained by conditioning probabilities on 'context information' beyond single nonterminals [12, 8]. Furthermore, it has been observed that conditioning on certain types of context information can be achieved by first translating context-free grammars to push-down automata, according to some parsing strategy, and then attaching probabilities to the transitions thereof [48, 38, 26]. More concretely, for some parsing strategies, the set of models that can be obtained by attaching probabilities to a push-down automaton constructed from a context-free grammar may include models that cannot be obtained by attaching probabilities to that grammar.

An implicit assumption of this methodology is that, conversely, any probabilistic model that can be obtained from a grammar can also be obtained from the associated push-down automaton, or in other words, the push-down automaton is at least as powerful as the grammar in terms of the set the potential models. If a parsing strategy does not satisfy this property, and if some potential models are lost in the mapping from the grammar to the push-down automaton, then this means that in some cases the strategy may lead to less rather than more accurate models. That LR parsing cannot be extended to become a probabilistic parsing strategy, as we mentioned above, means that the above property is not satisfied by this parsing strategy. This is contrary to what is suggested by some publications on probabilistic LR parsing, such as [5] and [19], which fail to observe that LR parsers may sometimes lead to less accurate models than the grammars from which they were constructed.

Some studies, such as [13, 9, 7, 52], propose lexicalized probabilistic context-free grammars, i.e., probabilistic models based on context-free grammars in which probabilities heavily rely on the terminal elements from input strings. Even if the current paper does not specifically deal with lexicalization, much of what we discuss pertains to lexicalized probabilistic context-free grammars as well.

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. Sections 6 and 7 provide examples of parsing strategies with and without the SPP. The examples without the SPP, most notably LR parsing, are shown not to be extendible to become probabilistic. A wider notion of extending a strategy to become probabilistic is provided by Section 8. We show that even under this wider notion, LR parsing cannot be extended to become probabilistic. Section 9 presents an application that concerns prefix probabilities. We end this paper with conclusions.

Some results reported here have appeared before in an abbreviated form in [33].

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 $\alpha \in (\Sigma \cup N)^*$. Without loss of generality, we assume that there is only one rule $S \rightarrow \sigma$ with the start symbol in the left-hand side, and furthermore that $\sigma \neq \epsilon$, where ϵ denotes the empty string.

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: $\alpha \overset{\pi}{\Rightarrow} \beta$ 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 $S \overset{\pi_1}{\Rightarrow} \cdots \overset{\pi_m}{\Rightarrow} \alpha$, 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 $S \overset{\pi_1}{\Rightarrow} \cdots \overset{\pi_m}{\Rightarrow} w$ 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 $A \overset{\pi_1}{\Rightarrow} \cdots \overset{\pi_m}{\Rightarrow} w$ 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 $\alpha \overset{\pi_1}{\Rightarrow} \cdots \overset{\pi_m}{\Rightarrow} \beta$, with $m \geq 0$ or $m > 0$, respectively. We say a CFG is *acyclic* if $A \Rightarrow^+ A$ does not hold for any $A \in N$.

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 [11], 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 $X \overset{x,y}{\mapsto} Y$ (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 $X \overset{x,y}{\mapsto} Y$, 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 $X \overset{\epsilon,\epsilon}{\mapsto} X_{push}$, $X \overset{\epsilon,\epsilon}{\mapsto} X_{pop}$ and $X \overset{\epsilon,\epsilon}{\mapsto} X_{swap}$. 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 [23, 2] 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 $(X_{init}, w_1, \epsilon) \overset{c}{\vdash^*} (\alpha, \epsilon, v_1)$, 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 $(\alpha, w_2, \epsilon) \vdash^* (X_{\mathit{final}}, \epsilon, v_2)$. 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 $\pi = A \to \alpha$.

For all nonterminals A, $\Sigma_{\pi = A \to \alpha}\ p_\mathcal{G}(\pi) = \Sigma_{\pi = A \to \alpha}\ \frac{n_\pi}{n_A} = \frac{n_A}{n_A} = 1$, which means that the PCFG $(\mathcal{G}, p_\mathcal{G})$ is proper. Furthermore, [11] 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 [4]. ■

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.*

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 $\mathcal{G} = (\Sigma_1, N, S, R)$, 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 $(1 \leq i \leq m)$:

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

[2] There is a property of push-down devices called *faiblement prédictif* (weakly predictive) [53]. Contrary to what this name may suggest however, this property is incomparable with the complement of our notion of SPP.

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 $\tau_i = Y_iX \mapsto Z_i$, $1 \leq i \leq m$. 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 $Y_i \mapsto Y_iX'$ 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 $\mathcal{G}$:

$$\begin{aligned} p_{\mathcal{A}}(c) \quad &= \quad \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_{X_{\mathit{init}}}}\ p_{\mathcal{G}}(\overline{v'})} && (3) \\ &= \quad \frac{p_{\mathcal{G}}(f(c))}{\Sigma_{v' \in L_{X_{\mathit{init}}}}\ p_{\mathcal{G}}(f(v'))} && (4) \\ &= \quad \frac{p_{\mathcal{G}}(f(c))}{\Sigma_{d \in D}\ p_{\mathcal{G}}(d)} && (5) \\ &= \quad p_{\mathcal{G}}(f(c)) && (6) \end{aligned}$$

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 that it can be considered as bijection from complete computations to complete derivations. Lastly we have 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:

$$p_{\mathcal{A}}(c') \quad = \quad \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_{Y_i}} \, p_{\mathcal{G}}(\overline{v'})} \tag{7}$$

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

$$\begin{aligned}
p_{\mathcal{A}}(c) \quad &= \quad p_{\mathcal{A}}(\tau_i) \cdot p_{\mathcal{A}}(c') && (8)\\
&= \quad \frac{\Sigma_{v' \in L_{\tau_i}} \, p_{\mathcal{G}}(\overline{v'})}{\Sigma_{v' \in L_X} \, p_{\mathcal{G}}(\overline{v'})} \cdot \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_{Y_i}} \, p_{\mathcal{G}}(\overline{v'})} && (9)\\
&= \quad \frac{p_{\mathcal{G}}(\overline{y_i}) \cdot \Sigma_{v' \in L_{Y_i}} \, p_{\mathcal{G}}(\overline{v'})}{\Sigma_{v' \in L_X} \, p_{\mathcal{G}}(\overline{v'})} \cdot \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_{Y_i}} \, p_{\mathcal{G}}(\overline{v'})} && (10)\\
&= \quad \frac{p_{\mathcal{G}}(\overline{y_i}) \cdot p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_X} \, p_{\mathcal{G}}(\overline{v'})} && (11)\\
&= \quad \frac{p_{\mathcal{G}}(\overline{y_i v})}{\Sigma_{v' \in L_X} \, p_{\mathcal{G}}(\overline{v'})} && (12)
\end{aligned}$$

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:

$$p_{\mathcal{A}}(c') \quad = \quad \frac{p_{\mathcal{G}}(\overline{v_1})}{\Sigma_{v'_1 \in L_{Y_i}} \, p_{\mathcal{G}}(\overline{v'_1})} \tag{13}$$

and

$$p_{\mathcal{A}}(c'') \quad = \quad \frac{p_{\mathcal{G}}(\overline{v_2})}{\Sigma_{v'_2 \in L_{X'_i}} \, p_{\mathcal{G}}(\overline{v'_2})} \tag{14}$$

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

$$p_{\mathcal{A}}(c) \quad = \quad p_{\mathcal{A}}(\tau_i) \cdot p_{\mathcal{A}}(c') \cdot p_{\mathcal{A}}(\tau) \cdot p_{\mathcal{A}}(c'') \tag{15}$$

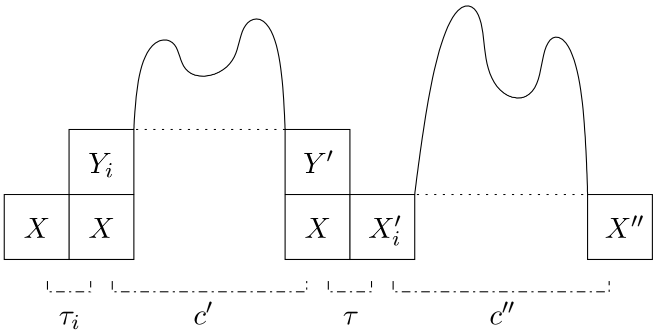

Figure 1: Development of the stack in the computation $c = \tau_i c' \tau c''$.

$$= \frac{\Sigma_{v' \in L_{\tau_i}} \; p_{\mathcal{G}}(\overline{v'})}{\Sigma_{v' \in L_X} \; p_{\mathcal{G}}(\overline{v'})} \cdot \frac{p_{\mathcal{G}}(\overline{v_1})}{\Sigma_{v_1' \in L_{Y_i}} \; p_{\mathcal{G}}(\overline{v_1'})} \cdot 1 \cdot \frac{p_{\mathcal{G}}(\overline{v_2})}{\Sigma_{v_2' \in L_{X_i'}} \; p_{\mathcal{G}}(\overline{v_2'})} \tag{16}$$

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

$$p_{\mathcal{A}}(c) = \frac{p_{\mathcal{G}}(\overline{v_1}) \cdot p_{\mathcal{G}}(\overline{v_2})}{\Sigma_{v' \in L_X} \; p_{\mathcal{G}}(\overline{v'})} \tag{17}$$

$$= \frac{p_{\mathcal{G}}(\overline{v_1 v_2})}{\Sigma_{v' \in L_X} \; p_{\mathcal{G}}(\overline{v'})} \tag{18}$$

$$= \frac{p_{\mathcal{G}}(\overline{v})}{\Sigma_{v' \in L_X} \; p_{\mathcal{G}}(\overline{v'})} \tag{19}$$

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 $\sum_{(A \to \alpha) \in R} |A\alpha|$, 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.

- $X \to YZ$ 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 $Y = X_{final}$.

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 $X_{final} \to \epsilon$, 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 [35, 24].

It has been shown by [15] that there is an infinite family of languages with the following property. The sizes of the smallest CFGs generating those languages are at least quadratically larger than the sizes of the smallest equivalent PDAs. Note that this increase in size cannot occur if PDAs satisfy both the CPP and the SPP, as we have shown above.

It is always possible to transform a PDA with the CPP but without the SPP to an equivalent PDA with both CPP and SPP, by a construction that increases the size of the PDA considerably (at least quadratically, in the light of the above construction and [15]). However, such transformations in general do not preserve parsing strategies and therefore are of minor interest to the issues discussed in this paper.

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 [4, 49]. 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 $X \rightarrow xY$ 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 [1, 10, 31], which allows an alternative proof of Theorem 4, as shown in [33]. 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 Section 7.

6 Parsing strategies with SPP

Many well-known parsing strategies with the CPP also have the SPP, such as top-down parsing [17], left-corner parsing [39] and PLR parsing [47], the first two of which we will define explicitly here, whereas of the third we will merely present a sketch. A fourth strategy that we will discuss is a combination of left-corner and top-down parsing, with special computational properties.

In order to simplify the presentation, we allow a new type of transition, without increasing the power of PDTs, viz. a combined push/swap transition of the form $X \overset{x,y}{\mapsto} XY$. Such a transition can be seen as short-hand for two transitions, the first of the form $X \mapsto XY_{x,y}$, where $Y_{x,y}$ is a new symbol not already in Q, and the second of the form $Y_{x,y} \overset{x,y}{\mapsto} Y$.

The first strategy we discuss is top-down parsing. For a fixed CFG grammar

$\mathcal{G} = (\Sigma, N, S, R)$, we define $\mathcal{S}_{TD}(\mathcal{G}) = (\mathcal{A}, f)$. Here $\mathcal{A} = (\Sigma, R, Q, [S \rightarrow \bullet\ \sigma], [S \rightarrow \sigma\ \bullet], \Delta)$, where $Q = \{[A \rightarrow \alpha \bullet \beta] \mid (A \rightarrow \alpha\beta) \in R\}$; these 'dotted rules' are well-known from [22, 14]. The transitions in Δ are:

- $[A \rightarrow \alpha \bullet a\beta] \overset{a,\epsilon}{\mapsto} [A \rightarrow \alpha a \bullet \beta]$ for each rule $A \rightarrow \alpha a\beta$;
- $[A \rightarrow \alpha \bullet B\beta] \overset{\epsilon,\pi}{\mapsto} [A \rightarrow \alpha \bullet B\beta]\ [B \rightarrow \bullet\ \gamma]$ for each pair of rules $A \rightarrow \alpha B\beta$ and $\pi = B \rightarrow \gamma$;
- $[A \rightarrow \alpha \bullet B\beta]\ [B \rightarrow \gamma\ \bullet] \mapsto [A \rightarrow \alpha B \bullet \beta]$.

The function f is the identity function on strings over R. If seen as a function on computations, then f is a bijection from complete computations of $\mathcal{A}$ to complete derivations of $\mathcal{G}$, as required by the definition of 'parsing strategy'.

If $\mathcal{G}$ is reduced, then $\mathcal{A}$ clearly has the CPP. That it also has the SPP can be argued as follows. Let us first remark that if $[A \rightarrow \alpha \bullet \beta] \rightsquigarrow X$ for some stack symbols $[A \rightarrow \alpha \bullet \beta]$ and X, then X must be of the form $[A \rightarrow \alpha\gamma \bullet \delta]$, for some γ and δ such that $\gamma\delta = \beta$. Now, if there are three transitions $X \mapsto XY$, $XY_1 \mapsto Z_1$ and $XY_2 \mapsto Z_2$ such that $Y \rightsquigarrow Y_1$ and $Y \rightsquigarrow Y_2$, then X must be of the form $[A \rightarrow \alpha \bullet B\beta]$ and Y of the form $[B \rightarrow \bullet\ \gamma]$ (strictly speaking $[B \rightarrow \bullet\ \gamma]_{\epsilon,\pi}$), Y_1 and Y_2 must both be $[B \rightarrow \gamma\ \bullet]$, and Z_1 and Z_2 must both be $[A \rightarrow \alpha B \bullet \beta]$. Hence the SPP is satisfied.

Since $\mathcal{S}_{TD}$ has both CPP and SPP, we may apply Theorem 4 to conclude that $\mathcal{S}_{TD}$ can be extended to become a probabilistic parsing strategy. A direct construction of a top-down PPDT from a PCFG $(\mathcal{G}, p_{\mathcal{G}})$ is obtained by extending the above construction such that probability 1 is assigned to all transitions produced by the first and third items, and probability $p_{\mathcal{G}}(\pi)$ is assigned to transitions produced by the second item.

The second strategy we discuss is left-corner (LC) parsing [39]. For a fixed CFG $\mathcal{G} = (\Sigma, N, S, R)$, we define the binary relation $\angle$ over $\Sigma \cup N$ by: $X \angle A$ if and only if there is an $\alpha \in (\Sigma \cup N)^*$ such that $(A \rightarrow X\alpha) \in R$, where $X \in \Sigma \cup N$. We define the binary relation $\angle^*$ to be the reflexive and transitive closure of $\angle$. This implies that $a \angle^* a$ for all $a \in \Sigma$.

We now define $\mathcal{S}_{LC}(\mathcal{G}) = (\mathcal{A}, f)$. Here $\mathcal{A} = (\Sigma, R \cup \{\dashv\}, Q, [S \rightarrow \bullet\ \sigma], [S \rightarrow \sigma\ \bullet], \Delta)$, where Q contains stack symbols of the form $[A \rightarrow \alpha \bullet \beta]$ where $(A \rightarrow \alpha\beta) \in R$ such that $\alpha \neq \epsilon \vee A = S$, and stack symbols of the form $[A \rightarrow \alpha \bullet Y\beta; X]$ where $(A \rightarrow \alpha Y\beta) \in R$ and $X, Y \in \Sigma \cup N$ such that $\alpha \neq \epsilon \vee A = S$ and $X \angle^* Y$. The latter type of stack symbol indicates that left corner X of goal Y in the right-hand side of rule $A \rightarrow \alpha Y\beta$ has just been recognized. The transitions in Δ are:

- $[A \rightarrow \alpha \bullet Y\beta] \overset{a,\epsilon}{\mapsto} [A \rightarrow \alpha \bullet Y\beta; a]$ for each rule $A \rightarrow \alpha Y\beta$ and $a \in \Sigma$ such that $\alpha \neq \epsilon \vee A = S$ and $a \angle^* Y$;
- $[A \rightarrow \alpha \bullet B\beta] \overset{\epsilon,\pi}{\mapsto} [A \rightarrow \alpha \bullet B\beta; C]$ for each pair of rules $A \rightarrow \alpha B\beta$ and $\pi = C \rightarrow \epsilon$ such that $\alpha \neq \epsilon \vee A = S$ and $C \angle^* B$;

- $[A \rightarrow \alpha \bullet B\beta; X] \overset{\epsilon,\pi}{\mapsto} [A \rightarrow \alpha \bullet B\beta; X]\ [C \rightarrow X \bullet \gamma]$ for each pair of rules $A \rightarrow \alpha B\beta$ and $\pi = C \rightarrow X\gamma$ such that $\alpha \neq \epsilon \vee A = S$ and $C \angle^* B$;

- $[A \rightarrow \alpha \bullet B\beta; X]\ [C \rightarrow X\gamma \bullet] \mapsto [A \rightarrow \alpha \bullet B\beta; C]$ for each pair of rules $A \rightarrow \alpha B\beta$ and $C \rightarrow X\gamma$ such that $\alpha \neq \epsilon \vee A = S$ and $C \angle^* B$;

- $[A \rightarrow \alpha \bullet Y\beta; Y] \overset{\epsilon,\dashv}{\mapsto} [A \rightarrow \alpha Y \bullet \beta]$ for each rule $A \rightarrow \alpha Y\beta$ such that $\alpha \neq \epsilon \vee A = S$.

The function f has to rearrange an output string to obtain a complete derivation. To make this possible, the output alphabet contains the symbol $\dashv$ in addition to rules from R. This symbol is used to mark the end of an upward path of nodes in the parse tree each of which, except the last, is the left-most daughter node of its mother node. As explained in [35], in the absence of such a symbol, it would be impossible to uniquely identify output strings with derivations of the input.[3]

The function f for the strategy $\mathcal{S}_{LC}$ is defined by Figure 2. Function f is defined in terms of function f_{LC}, which has two arguments. The first argument, d, is either the empty string or a subderivation that has already been constructed. The second argument is a suffix of the output string originally supplied as argument to f. Function f_{LC} removes the first symbol π from the output string, which will be a rule $A \rightarrow XX_1 \cdots X_l$ or $A \rightarrow \epsilon$. In the former case, d must be ϵ if $X \in \Sigma_1$ and d must be a subderivation from nonterminal X otherwise. The function is then called recursively zero or more times, once for each nonterminal in $X_1 \cdots X_l$, to obtain more subderivations d_i, $1 \leq i \leq l$, each of which is obtained by consuming a subsequent part of the output string. These subderivations are combined into a larger subderivation $d' = \pi d d_1 \cdots d_l$. Depending on the question whether we encounter $\dashv$ as the immediately following symbol of the output string, we return the derivation d' and the remainder v' of the output string, or call $\mathcal{S}_{LC}$ recursively once more to obtain a larger subderivation.

It can be easily shown that this strategy has the CPP. Regarding the SPP, note that if there are two transitions $[A \rightarrow \alpha \bullet B\beta; X] \overset{\epsilon,\pi}{\mapsto} [A \rightarrow \alpha \bullet B\beta; X]\ [C \rightarrow X \bullet \gamma]$ and $[A \rightarrow \alpha \bullet B\beta; X]\ Y_1 \mapsto Z_1$ such that $[C \rightarrow X \bullet \gamma] \rightsquigarrow Y_1$, then Y_1 must be $[C \rightarrow X\gamma \bullet]$ and Z_1 must be $[A \rightarrow \alpha \bullet B\beta; C]$, which means that Z_1 is uniquely determined by the first transition.

Since $\mathcal{S}_{LC}$ has both CPP and SPP, left-corner parsing can be extended to become a probabilistic parsing strategy. A direct construction of probabilistic left-corner parsers from PCFGs has been presented by [50].

Since at most two rules occur in each of the items above, the size of a (probabilistic) left-corner parser is $\mathcal{O}(|\mathcal{G}|^2)$, where $|\mathcal{G}|$ denotes the size of $\mathcal{G}$. This is the same complexity as that of the direct construction by [50]. This is in contrast to a construction of 'shift-reduce' PPDAs out of PCFGs from [1], which were of

[3] In [35, pp. 22–23] a context-free grammar is considered that consists of the set of rules $R = \{S \rightarrow aS, S \rightarrow Sb, S \rightarrow c\}$. It is shown that any left-corner push-down transducer using only R as output alphabet would output at most one string for each input string, whereas there may be several derivations of the input, as the grammar is ambiguous.

$$
\begin{array}{rcl}
f(v) & = & d \\
 & & \text{where} \\
 & & (d, \epsilon) = f_{LC}(\epsilon, v) \\
f_{LC}(d, \pi v_0) & = & (d'', v'') \\
 & & \text{where} \\
 & & l \text{ is such that } \pi = A \rightarrow X X_1 \cdots X_l \ \text{ or} \\
 & & \qquad \pi = A \rightarrow \epsilon \wedge l = 0 \\
 & & (d_1, v_1) = \text{if } X_1 \in \Sigma_1 \text{ then } (\epsilon, v_0) \text{ else } f_{LC}(\epsilon, v_0) \\
 & & \ldots \\
 & & (d_l, v_l) = \text{if } X_l \in \Sigma_1 \text{ then } (\epsilon, v_{l-1}) \text{ else } f_{LC}(\epsilon, v_{l-1}) \\
 & & d' = \pi d d_1 \cdots d_l \\
 & & (d'', v'') = \text{if } \neg v' = v_l \text{ then } (d', v') \text{ else } f_{LC}(d', v_l)
\end{array}
$$

Figure 2: Function f for $\mathcal{S}_{LC}$.

size $\mathcal{O}(|\mathcal{G}|^5)$.[4] The "conjecture that there is no *concise* translation of PCFGs into shift-reduce PPDAs" from [1] is made less significant by the earlier construction by [50] and our construction above. It must be noted however that the 'shift-reduce' model adhered to by [1] is more restrictive than the PDT models adhered to by [50] and by us.

When we look at upper bounds on the sizes of PPDAs (or PPDTs) that describe the same probability distributations as given PCFGs, and compare these with the upper bounds for (non-probabilistic) PDAs (or PDTs) for given CFGs, we can make the following observation. Theorem 3 states that parsing strategies without the CPP cannot be extended to become probabilistic. Furthermore, [25] has shown that for certain fixed languages the smallest PDAs without the CPP are much smaller than the smallest PDAs with the CPP. It may therefore appear that probabilistic PDAs are in general larger than non-probabilistic ones. However, the automata studied by [25] pertain to very specific languages, and at this point there is little reason to believe that the demonstrated results for these languages carry over to any reasonable strategy for *general* CFGs.

[4] This construction consisted of a transformation to Chomsky normal form followed by a transformation to Greibach normal form (GNF) [17]. Its worse-case time complexity, established in p.c. with David McAllester, is reached for a family of CFGs $(\mathcal{G}_n)_{n \geq 2}$, defined by $\mathcal{G}_n = (\{a_1, \ldots, a_n\}, \{A_1, \ldots, A_n\}, A_1, R)$, where R contains the rules $A_i \rightarrow A_{i+1}$, for $1 \leq i \leq n-1$, $A_n \rightarrow A_1$, and $A_i \rightarrow A_i\ A_i$ and $A_i \rightarrow a_i$, for $1 \leq i \leq n$. After transformation to GNF, the grammar contains n^5 rules of the form $A_{i_1}/A_{i_2} \rightarrow a_{i_3}\ A_{i_2}/A_{i_4}\ A_{i_1}/A_{i_5}$, with $1 \leq i_1, i_2, i_3, i_4, i_5 \leq n$. In [3] a more economical transformation to Greibach normal form is given; straightforward extension to probabilities leads to probabilistic parsers of the type considered by [1] of size $\mathcal{O}(|\mathcal{G}|^4)$. An older transformation of PCFGs to GNF, in [18], yields grammars of exponential size.

The third parsing strategy that we discuss is PLR parsing [47]. Since it is very similar to LC parsing, we merely provide a sketch. The stack symbols for PLR parsing are like those for LC parsing, except that the parts of rules following the dot are omitted. Thus, instead of symbols of the form $[A \rightarrow \alpha \bullet \beta]$ and of the form $[A \rightarrow \alpha \bullet \beta; X]$, a PLR parser manipulates stack symbols $[A \rightarrow \alpha]$ and $[A \rightarrow \alpha; X]$, respectively. That β is omitted means that PLR parsers may postpone commitment to one from two similar rules $A \rightarrow \alpha\beta$ and $A \rightarrow \alpha\beta'$ until the point is reached where β and β' differ. In this sense PLR parsing is less predictive than LC parsing, although it still satisfies the strong predictiveness property, so that it can be extended to become probabilistic.

There are two minor differences between the transitions of LC parsers and those of PLR parsers. The first is the simplification of stack symbols as explained above. The second is that for PLR, output of a rule is delayed until it is completely recognized. The resulting output strings are right-most derivations in reverse, which requires different functions f than in the case of LC parsing. Note that right-most derivations can be effectively mapped to corresponding parse trees, and parse trees can be effectively mapped to corresponding left-most derivations. Hence the required functions f clearly exist.

The last strategy to be discussed in this section is a combination of left-corner and top-down parsing. It has the special property that, provided the fixed CFG is acyclic, the length of computations is bounded by a linear function on the length of the input, which means that the parser cannot ‘loop’ on any input. Note that if the grammar is not acyclic, computations of unbounded length cannot be avoided by any parsing strategy. From this perspective, this parsing strategy, which we will call *ϵ-LC* parsing, is optimal. It is based on [28], and a related idea for LR parsing was described by [30]. The special termination properties of this strategy will be needed in Section 9.

We first define the binary relation $\angle_\epsilon$ over $\Sigma \cup N$ by: $X \angle_\epsilon A$ if and only if there are $\alpha, \beta \in (\Sigma \cup N)^*$ such that $(A \rightarrow \alpha X \beta) \in R$ and $\alpha \Rightarrow^* \epsilon$. Relation $\angle_\epsilon$ differs from the relation $\angle$ defined earlier in that epsilon-generating nonterminals at the beginning of a rule may be ignored.

The stack symbols are now of the form $[A \rightarrow \alpha \bullet \beta, \mu \bullet \nu]$ or of the form $[A \rightarrow \alpha \bullet Y\beta, \mu \bullet \nu; X]$. Similar to the stack symbols for pure LC parsing, we have $\alpha \neq \epsilon \vee A = S$ and $X \angle_\epsilon^* Y$. Different is the additional dotted expression $\mu \bullet \nu$, which is such that $\mu\nu$ is a string of epsilon-generating nonterminals, occurring at the beginning of the right-hand side of a rule $A \rightarrow \mu\nu\alpha\beta$ or $A \rightarrow \mu\nu\alpha Y \beta$, respectively. The string $\mu\nu$ will be ignored in the part of the strategy that behaves like left-corner parsing, where $\mu = \epsilon$. However, when the dot of the first dotted expression is at the end, i.e., when we obtain a stack symbol of the form $[A \rightarrow \alpha \bullet, \bullet \nu]$, then top-down parsing will be activated to retrieve epsilon-generating subderivations for the nonterminals in ν, and the dot will move through ν from left to right.[5]

[5] Although such subderivations can also be pre-compiled during construction of the PDT, we refrain from doing so since this could lead to a PDT of exponential size.

We have $X_{init} = [S \rightarrow \bullet\ \sigma, \bullet]$ and $X_{final} = [S \rightarrow \sigma\ \bullet, \bullet]$, where for technical reasons, and without loss of generality, we assume that σ does not contain any epsilon-generating nonterminals. Next to the symbols from R and the symbol $\dashv$, the output alphabet Σ_2 also includes the set of integers $\{0, \ldots, l-1\}$, where $l = |\alpha|$ for a rule $(A \rightarrow \alpha) \in R$ of maximal length; the purpose of such integers will become clear below. For the definition of the set of transitions, we will be less precise than for $\mathcal{S}_{TD}$ and $\mathcal{S}_{LC}$, to prevent cluttering up the presentation with details. We point out however that in order to produce a reduced PDT from a reduced CFG, further side conditions are needed for all items below:

- $[A \rightarrow \alpha \bullet Y\beta, \bullet\ \mu] \overset{a,\epsilon}{\mapsto} [A \rightarrow \alpha \bullet Y\beta, \bullet\ \mu; a]$ for $a \in \Sigma$ such that $a \angle_\epsilon^* Y$;
- $[A \rightarrow \alpha \bullet B\beta, \bullet\ \mu] \overset{\epsilon,\pi 0}{\mapsto} [A \rightarrow \alpha \bullet B\beta, \bullet\ \mu; C]$ for $\pi = C \rightarrow \epsilon$ such that $C \angle^* B$;
- $[A \rightarrow \alpha \bullet B\beta, \bullet\ \mu; X] \overset{\epsilon,\pi m}{\mapsto} [A \rightarrow \alpha \bullet B\beta, \bullet\ \mu; X]\ [C \rightarrow X \bullet \gamma, \bullet\ \mu']$ for $\pi = C \rightarrow \mu' X \gamma$ such that $C \angle_\epsilon^* B$ and $\mu' \Rightarrow^* \epsilon$, where $m = |\mu'|$;
- $[A \rightarrow \alpha \bullet B\beta, \bullet\ \mu; X]\ [C \rightarrow X\gamma\ \bullet, \mu'\ \bullet] \mapsto [A \rightarrow \alpha \bullet B\beta, \bullet\ \mu; C]$;
- $[A \rightarrow \alpha \bullet Y\beta, \bullet\ \mu; Y] \overset{\epsilon,\dashv}{\mapsto} [A \rightarrow \alpha Y \bullet \beta, \bullet\ \mu]$;
- $[A \rightarrow \alpha\ \bullet, \mu \bullet B\nu] \overset{\epsilon,\pi}{\mapsto} [A \rightarrow \alpha\ \bullet, \mu \bullet B\nu]\ [B \rightarrow \bullet, \bullet\ \mu']$ for $\pi = B \rightarrow \mu'$ such that $\mu' \Rightarrow^* \epsilon$;
- $[A \rightarrow \alpha\ \bullet, \mu \bullet B\nu]\ [B \rightarrow \bullet, \mu'\ \bullet] \mapsto [A \rightarrow \alpha\ \bullet, \mu B \bullet \nu]$.

The first five items are almost identical to the five items we presented for $\mathcal{S}_{LC}$, except that strings μ of epsilon-generating nonterminals at the beginning of rules are ignored. The length m of a string μ is output just after the relevant grammar rule is output, in the second and third items. This length m will be needed to define function f below.

The last two items follow a top-down strategy, but only for epsilon-generating rules. The produced transitions do what was deferred by the left-corner part of the strategy: they construct subderivations for the epsilon-generating nonterminals in strings μ.

The function f, which produces a complete derivation from an output string, is defined through two auxiliary functions, viz. $f_{\epsilon\text{-}LC}$ for the left-corner part and $f_{\epsilon\text{-}TD}$ for the top-down part, as shown in Figure 3.

The function $f_{\epsilon\text{-}LC}$ is similar to f_{LC} defined in Figure 2. The main difference is that now subderivations deriving ϵ for the first m nonterminals in the right-hand side of a rule are obtained by calls of the function $f_{\epsilon\text{-}TD}$. For a suffix v of an output string, $f_{\epsilon\text{-}TD}(v)$ yields a pair $(\pi d_1 \cdots d_l, v_l)$ such that $v = \pi d_1 d_2 \cdots d_l v_l$. In other words, $f_{\epsilon\text{-}TD}$ does nothing more than split its argument into two parts. The length of the first part $\pi d_1 \cdots d_l$ depends on the length l of the right-hand side of rule π and on the lengths of right-hand sides of rules that are visited recursively.

It can be easily seen that $\mathcal{S}_{\epsilon\text{-}LC}$ has both CPP and SPP. The size of a produced PDT is now $\mathcal{O}(|\mathcal{G}|^3)$, rather than $\mathcal{O}(|\mathcal{G}|^2)$ as in the case of $\mathcal{S}_{LC}$.

$$
\begin{aligned}
f(v) &= d \\
&\quad \text{where} \\
&\quad (d,\epsilon) = f_{\epsilon\text{-}LC}(\epsilon, v) \\
f_{\epsilon\text{-}LC}(d, \pi m v_0) &= (d'', v'') \\
&\quad \text{where} \\
&\quad l \text{ is such that } \pi = A \rightarrow B_1 \cdots B_m X X_1 \cdots X_l \ \text{ or} \\
&\qquad\quad \pi = A \rightarrow \epsilon \wedge l = 0 \\
&\quad (d_1, v_1) = \text{if } X_1 \in \Sigma_1 \text{ then } (\epsilon, v_0) \text{ else } f_{\epsilon\text{-}LC}(\epsilon, v_0) \\
&\quad \ldots \\
&\quad (d_l, v_l) = \text{if } X_l \in \Sigma_1 \text{ then } (\epsilon, v_{l-1}) \text{ else } f_{\epsilon\text{-}LC}(\epsilon, v_{l-1}) \\
&\quad (d'_1, v_{l+1}) = f_{\epsilon\text{-}TD}(v_l) \\
&\quad \ldots \\
&\quad (d'_m, v_{l+m}) = f_{\epsilon\text{-}TD}(v_{l+m-1}) \\
&\quad d' = \pi d'_1 \cdots d'_m d d_1 \cdots d_l \\
&\quad (d'', v'') = \text{if } \dashv v' = v_{l+m} \text{ then } (d', v') \text{ else } f_{\epsilon\text{-}LC}(d', v_{l+m}) \\
f_{\epsilon\text{-}TD}(v) &= (\pi d_1 \cdots d_l, v_l) \\
&\quad \text{where} \\
&\quad \pi v_0 = v \\
&\quad l \text{ is such that } \pi = A \rightarrow B_1 \cdots B_l \\
&\quad (d_1, v_1) = f_{\epsilon\text{-}TD}(v_0) \\
&\quad \ldots \\
&\quad (d_l, v_l) = f_{\epsilon\text{-}TD}(v_{l-1})
\end{aligned}
$$

Figure 3: Function f for $\mathcal{S}_{\epsilon\text{-}LC}$.

7 Parsing strategies without SPP

In this section we show that the absence of the strong predictiveness property may mean that a parsing strategy with the CPP cannot be extended to become a probabilistic parsing strategy. We first illustrate this for LR(0) parsing, formalized as a parsing strategy $\mathcal{S}_{LR}$, which has the CPP but not the SPP, as we will see. We assume the reader is familiar with LR parsing; see [46].

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

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

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:

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

(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. $\{C \rightarrow x \bullet c, D \rightarrow x \bullet d\}$.

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(0) 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. [5] and [19], 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, [48] has 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. In addition, [32] has shown that the accuracy can even be improved by non-standard probabilistic LR parsers that lack the properness condition.

By way of Theorem 4, it follows indirectly from the above that LR parsing lacks the SPP. For the somewhat simpler ELR(0) parsing strategy, to be discussed next, we will give a direct explanation of why it lacks the SPP. A direct explanation for LR parsing is much more involved and therefore is not reported here, although the argument is essentially of the same nature as the one we discuss for ELR parsing.

ELR parsing is not as well-known as LR parsing. It was originally formulated as a type of parsing strategy for extended CFGs [37, 24], but its restriction to normal CFGs is interesting in its own right, as argued by [29]. ELR parsing for CFGs is also related to the tabular algorithm from [54].

Concerning the representation of right-hand sides of rules, stack symbols for ELR parsing are similar to those for PLR parsing: only the part of a right-hand side is represented that consists of the grammar symbols that have been processed. Different from LC and PLR parsing is however that a stack symbol for ELR parsing contains a set consisting of one or more nonterminals from the left-hand sides of pairwise similar rules, rather than a single such nonterminal. This allows the commitment to certain rules, and in particular to their left-hand sides, to be postponed even longer than for LC and PLR parsing.

Thus, for a given CFG $\mathcal{G} = (\Sigma,\ N,\ S,\ R)$, we construct a pair $\mathcal{S}_{ELR}(\mathcal{G}) = (\mathcal{A}, f)$. Here $\mathcal{A} = (\Sigma,\ R,\ Q,\ [\{S\} \rightarrow \epsilon],\ [\{S\} \rightarrow \sigma],\ \Delta)$, where Q is a subset of $\{[\Gamma \rightarrow \alpha] \mid \Gamma \subseteq N \wedge \forall A \in \Gamma \exists \beta[(A \rightarrow \alpha\beta) \in R]\} \cup \{[\Gamma \rightarrow \alpha; B] \mid \Gamma \subseteq N \wedge \forall A \in \Gamma \exists \beta[(A \rightarrow \alpha\beta) \in R \wedge B \in N]\}$.

We provide simultaneous inductive definitions of Q and Δ:

- $[\{S\} \rightarrow \epsilon] \in Q$;
- For $[\Gamma \rightarrow \alpha] \in Q$, rule $A \rightarrow \alpha Y \beta$ and $a \in \Sigma$ such that $A \in \Gamma$ and $a \angle^* Y$, let $[\Gamma \rightarrow \alpha; a] \in Q$ and $[\Gamma \rightarrow \alpha] \overset{a,\epsilon}{\mapsto} [\Gamma \rightarrow \alpha; a] \in \Delta$;
- For $[\Gamma \rightarrow \alpha] \in Q$, rules $A \rightarrow \alpha B \beta$ and $\pi = C \rightarrow \epsilon$ such that $A \in \Gamma$ and $C \angle^* B$, let $[\Gamma \rightarrow \alpha; C] \in Q$ and $[\Gamma \rightarrow \alpha] \overset{\epsilon,\pi}{\mapsto} [\Gamma \rightarrow \alpha; C] \in \Delta$;
- For $[\Gamma_1 \rightarrow \alpha; X] \in Q$ and $\Gamma_2 = \{C \mid \exists (A \rightarrow \alpha B \beta) \in R[A \in \Gamma_1 \wedge C \rightarrow X\gamma \wedge C \angle^* B]\} \neq \emptyset$, let $[\Gamma_2 \rightarrow X] \in Q$ and $[\Gamma_1 \rightarrow \alpha; X] \mapsto [\Gamma_1 \rightarrow \alpha; X]\ [\Gamma_2 \rightarrow X] \in \Delta$;
- For $[\Gamma_1 \rightarrow \alpha; X], [\Gamma_2 \rightarrow X\gamma] \in Q$, rules $A \rightarrow \alpha B \beta$ and $\pi = C \rightarrow X\gamma$ such that $A \in \Gamma_1$, $C \in \Gamma_2$ and $C \angle^* B$, let $[\Gamma_1 \rightarrow \alpha; C] \in Q$ and $[\Gamma_1 \rightarrow \alpha; X]\ [\Gamma_2 \rightarrow X\gamma] \overset{\epsilon,\pi}{\mapsto} [\Gamma_1 \rightarrow \alpha; C] \in \Delta$;
- For $[\Gamma_1 \rightarrow \alpha; Y] \in Q$ and $\Gamma_2 = \{A \in \Gamma_1 \mid \exists \beta[(A \rightarrow \alpha Y \beta) \in R]\} \neq \emptyset$, let $[\Gamma_2 \rightarrow \alpha Y] \in Q$ and $[\Gamma_1 \rightarrow \alpha; Y] \mapsto [\Gamma_2 \rightarrow \alpha Y] \in \Delta$.

Note that the last five items are very similar to the five items for LC parsing. In the second last item, we have assumed the availability of combined pop/swap transitions of the form $XY \overset{x,y}{\mapsto} Z$. Such a transition can be seen as short-hand

$$
\begin{array}{rcl}
 & & [\{S\} \to \epsilon] \overset{a,\epsilon}{\mapsto} [\{S\} \to \epsilon; a] \\
 & & [\{S\} \to \epsilon; a] \mapsto [\{S\} \to \epsilon; a]\ [\{A\} \to a] \\
 & & [\{A\} \to a] \overset{x,\epsilon}{\mapsto} [\{A\} \to a; x] \\
 & & [\{A\} \to a; x] \mapsto [\{A\} \to a; x]\ [\{C, D\} \to x] \\
\tau_c & = & [\{C, D\} \to x] \overset{c,\epsilon}{\mapsto} [\{C, D\} \to x; c] \\
\tau_d & = & [\{C, D\} \to x] \overset{d,\epsilon}{\mapsto} [\{C, D\} \to x; d] \\
 & & [\{C, D\} \to x; c] \mapsto [\{C\} \to xc] \\
 & & [\{A\} \to a; x]\ [\{C\} \to xc] \overset{\epsilon,\pi_C}{\mapsto} [\{A\} \to a; C] \\
 & & [\{A\} \to a; C] \mapsto [\{A\} \to aC] \\
 & & [\{S\} \to \epsilon; a]\ [\{A\} \to aC] \overset{\epsilon,\pi_{A_1}}{\mapsto} [\{S\} \to \epsilon; A] \\
 & & [\{C, D\} \to x; d] \mapsto [\{D\} \to xd] \\
 & & [\{A\} \to a; x]\ [\{D\} \to xd] \overset{\epsilon,\pi_D}{\mapsto} [\{A\} \to a; D] \\
 & & [\{A\} \to a; D] \mapsto [\{A\} \to aD] \\
 & & [\{S\} \to \epsilon; a]\ [\{A\} \to aD] \overset{\epsilon,\pi_{A_2}}{\mapsto} [\{S\} \to \epsilon; A] \\
 & & [\{S\} \to \epsilon; A] \mapsto [\{S\} \to A] \\
 & & [\{S\} \to A] \overset{b,\epsilon}{\mapsto} [\{S\} \to A; b] \\
 & & [\{S\} \to A; b] \mapsto [\{S\} \to A; b]\ [\{B\} \to b] \\
 & & [\{B\} \to b] \overset{x,\epsilon}{\mapsto} [\{B\} \to b; x] \\
 & & [\{B\} \to b; x] \mapsto [\{B\} \to b; x]\ [\{C, D\} \to x] \\
 & & [\{B\} \to b; x]\ [\{C\} \to xc] \overset{\epsilon,\pi_C}{\mapsto} [\{B\} \to b; C] \\
 & & [\{B\} \to b; C] \mapsto [\{B\} \to bC] \\
 & & [\{S\} \to A; b]\ [\{B\} \to bC] \overset{\epsilon,\pi_{B_1}}{\mapsto} [\{S\} \to A; B] \\
 & & [\{B\} \to b; x]\ [\{D\} \to xd] \overset{\epsilon,\pi_D}{\mapsto} [\{B\} \to b; D] \\
 & & [\{B\} \to b; D] \mapsto [\{B\} \to bD] \\
 & & [\{S\} \to A; b]\ [\{B\} \to bD] \overset{\epsilon,\pi_{B_2}}{\mapsto} [\{S\} \to A; B] \\
 & & [\{S\} \to A; B] \mapsto [\{S\} \to AB]
\end{array}
$$

Figure 4: Transitions for the ELR(0) parsing strategy.

for two transitions, the first of the form $XY \mapsto Z_{x,y}$, where $Z_{x,y}$ is a new symbol not already in Q, and the second of the form $Z_{x,y} \overset{x,y}{\mapsto} Z$.

The function f is defined as in the case of PLR parsing, and turns a complete right-most derivation in reverse into a complete derivation.

ELR parsing has the CPP but, like LR parsing, it lacks the SPP. The problem is caused by transitions of the form $[\mathit{\Gamma}_1 \to \alpha; X]\ [\mathit{\Gamma}_2 \to X\gamma] \overset{\epsilon,\pi}{\mapsto} [\mathit{\Gamma}_1 \to \alpha; C]$. Intuitively, a subcomputation that recognizes γ, directly after recognition of X, only commits to a choice of the left-hand side nonterminal C from $\mathit{\Gamma}_2$ after γ has been completely recognized, and this choice is communicated to lower areas of the stack through this pop transition.

That ELR parsing can indeed not be extended to a probabilistic parsing strategy can be shown by considering the same CFG as above. From the set of transitions, shown in Figure 4, we restrict our attention to the following two:

$$\tau_c = [\{C, D\} \to x] \overset{c,\epsilon}{\mapsto} [\{C, D\} \to x; c]$$
$$\tau_d = [\{C, D\} \to x] \overset{d,\epsilon}{\mapsto} [\{C, D\} \to x; d]$$

This is the only pair of transitions that can be applied for one and the same top-of-stack. The rest of the proof is identical to that in the case of LR parsing.

Problems with the extension of ELR parsing to become a probabilistic parsing strategy have been pointed out before by [51], who furthermore proposed an alternative type of probabilistic push-down automaton that is capable of computing multiple probabilities for each subderivation. However, since a transition of such an automaton may perform an unbounded number of elementary computations on probabilities, we feel this automaton model cannot realistically express the behaviour of probabilistic parsers, and therefore it will not be considered further here.

8 Extension in the wide sense

The main result from the previous section is that, in general, there is no construction of probabilistic LR parsers from PCFGs such that, firstly, a probabilistic LR parser has the same set of transitions as the LR parser that would be constructed from the CFG in the non-probabilistic case and, secondly, the probabilistic LR parser has the same probability distribution as the given PCFG.

There is a construction proposed by [56, 55, 34] that operates under different assumptions. In particular, a probabilistic LR parser constructed from a certain PCFG may possess several 'copies' of one and the same LR state from the (non-probabilistic) LR parser constructed from the CFG, each annotated with some additional information to distinguish it from other copies of the same LR state. Each such copy behaves as the corresponding LR state from the LR parser if we neglect probabilities. Transitions may however obtain different probabilities if they operate on different copies of identical LR states, based on the additional information attached to the LR states.

By this construction, there are many PCFGs for which one may obtain a probabilistic LR parser that describes the same probability distribution. This even holds for the PCFG we discussed in the previous section, although we have shown that a probabilistic LR parser *without* an extended LR state set could not describe the same probability distribution. A serious problem with this approach is however that the required number of copies of each LR state is potentially infinite.

In this section we formulate these observations in terms of general parsing strategies and a wider notion of extension to probabilistic parsing strategies. We also show that the above-mentioned problem with infinite numbers of states is inherent in LR parsing, rather than due to the particular construction of LR parsers from PCFGs by [56, 55, 34].

We first introduce some auxiliary notation and terminology. Let $\mathcal{A}$ and $\mathcal{A}'$ be two PDTs and let g be a function mapping the stack symbols of $\mathcal{A}'$ to the stack symbols of $\mathcal{A}$. If τ is a transition of the form $X \mapsto XY$, $YX \mapsto Z$ or $X \overset{x,y}{\mapsto} Y$ from $\mathcal{A}'$, then we let $g(\tau)$ denote a transition of the form $g(X) \mapsto g(X)g(Y)$, $g(Y)g(X) \mapsto g(Z)$ or $g(X) \overset{x,y}{\mapsto} g(Y)$, respectively. This effectively extends g to a function from transitions to transitions. Note that a transition $g(\tau)$ may, but need not be a transition from $\mathcal{A}$. In the same vein, we extend g to a function from computations of $\mathcal{A}'$ to sequences of transitions (which may, but need not be computations of $\mathcal{A}$), by applying g element-wise as a function on transitions.

For PDTs $\mathcal{A} = (\Sigma_1, \Sigma_2, Q, X_{init}, X_{final}, \Delta)$ and $\mathcal{A}' = (\Sigma'_1, \Sigma'_2, Q', X'_{init}, X'_{final}, \Delta')$, we say $\mathcal{A}'$ is an *expansion* of $\mathcal{A}$ if $\Sigma'_1 = \Sigma_1$, $\Sigma'_2 = \Sigma_2$ and there is a function g such that:

- g is a surjective function from Q' to Q.
- Extended to transitions, g is a surjective function from Δ' to Δ.
- Extended to computations, g is a bijective function from the set of computations of $\mathcal{A}'$ to the set of computations of $\mathcal{A}$.

In other words, for each stack symbol from Q, Q' may contain one or more corresponding stack symbols. The language that is accepted and the output strings that are produced for given input strings remain the same however. Furthermore, that g is a bijection on computations implies that the behaviour of the two automata is identical in terms of e.g. the length of computations and the amount of nondeterminism encountered within those computations.

To illustrate these definitions, assume we have an arbitrary PDT $\mathcal{A}$. We construct a second PDT $\mathcal{A}'$ that is an expansion of $\mathcal{A}$. It has the same input and output alphabets, and for each stack symbol X from $\mathcal{A}$, $\mathcal{A}'$ has two stack symbols $(X, 0)$ and $(X, 1)$. A second component 0 signifies that the distance of the stack symbol to the bottom of the stack is even, and 1 that it is odd. Naturally, if X_{init} and X_{final} are the initial and final stack symbols of $\mathcal{A}$, we choose the initial and final stack symbols of $\mathcal{A}'$ to be $(X_{init}, 0)$ and $(X_{final}, 0)$, as they have distance 0 to the bottom of the stack. For each transition of the form $X \mapsto XY$, $YX \mapsto Z$ or $X \overset{x,y}{\mapsto} Y$ from $\mathcal{A}$, we let $\mathcal{A}'$ have the transitions $(X, i) \mapsto (X, i)(Y, 1 - i)$, $(Y, i)(X, 1 - i) \mapsto (Z, i)$ or $(X, i) \overset{x,y}{\mapsto} (Y, i)$, respectively, for both $i = 0$ and $i = 1$. Obviously, the function g mapping stack symbols from $\mathcal{A}'$ to stack symbols from $\mathcal{A}$ is given by $g((X, i)) = X$ for all X and $i \in \{0, 1\}$.

We now come to the central definition of this section. We say that probabilistic parsing strategy $\mathcal{S}'$ is an *extension in the wide sense* 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 $\mathcal{A}'$ that is an expansion of $\mathcal{A}$ and some $p_{\mathcal{A}'}$. This definition allows more probabilistic parsing strategies $\mathcal{S}'$ to be related to a given strategy $\mathcal{S}$ than the definition of extension from Section 3.

LR parsing however, which we know can not be extended to a probabilistic strategy in the narrow sense from Section 3, can neither be extended in the wide sense to a probabilistic parsing strategy. To prove this, consider the following PCFG $(\mathcal{G}, p_{\mathcal{G}})$, taken from [56] with minor modifications:

$$\begin{array}{ll} \pi_S \ = S \rightarrow A, & p_{\mathcal{G}}(\pi_S) = 1 \\ \pi_{A_1} = A \rightarrow B, & p_{\mathcal{G}}(\pi_{A_1}) = \frac{1}{2} \\ \pi_{A_2} = A \rightarrow C, & p_{\mathcal{G}}(\pi_{A_2}) = \frac{1}{2} \\ \pi_{B_1} = B \rightarrow aB, & p_{\mathcal{G}}(\pi_{B_1}) = \frac{1}{3} \\ \pi_{B_2} = B \rightarrow b, & p_{\mathcal{G}}(\pi_{B_2}) = \frac{2}{3} \\ \pi_{C_1} = C \rightarrow aC, & p_{\mathcal{G}}(\pi_{C_1}) = \frac{2}{3} \\ \pi_{C_2} = C \rightarrow c, & p_{\mathcal{G}}(\pi_{C_2}) = \frac{1}{3} \end{array}$$

The CFG $\mathcal{G}$ generates strings of the form $a^n b$ and $a^n c$ for any $n \geq 0$. Observe that $\frac{p_{\mathcal{G}}(a^n b)}{p_{\mathcal{G}}(a^n c)} = \frac{\frac{1}{2}\cdot\left(\frac{1}{3}\right)^n\cdot\frac{2}{3}}{\frac{1}{2}\cdot\left(\frac{2}{3}\right)^n\cdot\frac{1}{3}} = \left(\frac{1}{2}\right)^{n-1}$.

Let $\mathcal{A}$ be such that $\mathcal{S}_{LR}(\mathcal{G}) = (\mathcal{A}, f)$ and consider input strings of the form $a^n b$ and $a^n c$, $n \geq 1$. After scanning the first n symbols, $\mathcal{A}$ reaches a configuration where the top-of-stack X is given by the set of (kernel) items:

$$X = \{B \rightarrow a \bullet B, C \rightarrow a \bullet C\}$$

There are three applicable transitions, representing shift actions over a, b and c, given by:

$$\begin{aligned} \tau_a &= X \overset{a,\epsilon}{\mapsto} X\ X \\ \tau_b &= X \overset{b,\epsilon}{\mapsto} X\ \{B \rightarrow b\ \bullet\} \\ \tau_c &= X \overset{c,\epsilon}{\mapsto} X\ \{C \rightarrow c\ \bullet\} \end{aligned}$$

After reading b or c, the remaining transitions are fully deterministic.

For a PDT $\mathcal{A}'$ that is an expansion of $\mathcal{A}$, we may have different stack symbols that are all mapped to X by function g. These stack symbols can be referred to as X_n, which occur as top-of-stack after scanning the first n symbols of $a^n b$ or $a^n c$, $n \geq 1$. We refer to the applicable transitions with top-of-stack X_n as:

$$\begin{aligned} \tau_{a,n} &= X_n \overset{a,\epsilon}{\mapsto} X_n\ X_{n+1} \\ \tau_{b,n} &= X_n \overset{b,\epsilon}{\mapsto} X_n\ \{B \rightarrow b\ \bullet\}_n \\ \tau_{c,n} &= X_n \overset{c,\epsilon}{\mapsto} X_n\ \{C \rightarrow c\ \bullet\}_n \end{aligned}$$

for certain stack symbols $\{B \rightarrow b\ \bullet\}_n$ and $\{C \rightarrow c\ \bullet\}_n$ that g maps to $\{B \rightarrow b\ \bullet\}$ and $\{C \rightarrow c\ \bullet\}$, respectively.

Now let us assume we have a probability function $p_{\mathcal{A}'}$ such that $(\mathcal{A}', p_{\mathcal{A}'})$ is a PPDT. Since the application of either $\tau_{b,n}$ or $\tau_{c,n}$ is the only nondeterministic step that distinguishes recognition of $a^n b$ from recognition of $a^n c$, $n \geq 1$, it follows that $\frac{p_{\mathcal{A}}(a^n b)}{p_{\mathcal{A}}(a^n c)} = \frac{p_{\mathcal{A}}(\tau_{b,n})}{p_{\mathcal{A}}(\tau_{c,n})}$. If $(\mathcal{A}', p_{\mathcal{A}'})$ assigns the same probabilities to strings over alphabet $\{a, b, c\}$ as $(\mathcal{G}, p_{\mathcal{G}})$, then $\frac{p_{\mathcal{A}}(\tau_{b,n})}{p_{\mathcal{A}}(\tau_{c,n})}$ must be equal to $\frac{p_{\mathcal{G}}(a^n b)}{p_{\mathcal{G}}(a^n c)} = \left(\frac{1}{2}\right)^{n-1}$ for each $n \geq 1$. Since $\left(\frac{1}{2}\right)^{n-1}$ is a different value for each n however, this would require $\mathcal{A}'$ to possess infinitely many stack symbols, which is in conflict with the definition of push-down transducers.

This shows that no probability function $p_{\mathcal{A}'}$ exists for any expansion $\mathcal{A}'$ of $\mathcal{A}$ such that $(\mathcal{A}', p_{\mathcal{A}'})$ assigns the same probabilities to strings over the alphabet as $(\mathcal{G}, p_{\mathcal{G}})$, and therefore LR parsing cannot be extended in the wide sense to become a probabilistic parsing strategy. With only minor changes to the proof, the same can be shown for ELR parsing.

9 Prefix probabilities

In this section we show that the behaviour of PPDTs on input can be simulated by dynamic programming. We also show how dynamic programming can be used for computing prefix probabilities. Prefix probabilities have important applications, e.g. in the area of speech recognition.

Our algorithm is a minor extension of an application of dynamic programming developed for non-probabilistic PDTs by [23, 2], and the treatment of probabilities is derived from [49].

Assume a fixed PPDT $(\mathcal{A}, p_{\mathcal{A}})$ and a fixed input string $a_1 \cdots a_n$. Consider a computation of the form $c_1 \tau c_2$, where $(X_{init}, a_1 \cdots a_i, \epsilon) \overset{c_1}{\vdash^*} (\alpha X, \epsilon, v_1)$, τ is of the form $X \mapsto XY'$, and $(Y', a_{i+1} \cdots a_j, \epsilon) \overset{c_2}{\vdash^*} (Y, \epsilon, v_2)$, for some stack symbols X, Y', Y, some input positions i and j $(0 \leq i \leq j \leq n)$, and some output strings v_1 and v_2. In words, the computation obtains top-of-stack X after scanning of a_i but before scanning of a_{i+1}, then applies a push transition, and then possibly further push, scan and pop transitions, which leads to Y on top of X after scanning of a_j but before scanning of a_{j+1}.

We now abstract away from some details of such a computation by just recording X, Y, i, j and its probability $p_1 = p_{\mathcal{A}}(c_1 \tau c_2)$. The probability p_1 is related to what is commonly called a *forward* probability, as it expresses the probability of the computation from the beginning onward.[6] The existence of the above computation is represented by an object that we will call a *table item*, written as $p_1 : \mathit{forward}(X, Y, i, j)$.

Similarly, consider a subcomputation of the form τc_2, where as before τ is of the form $X \mapsto XY'$, and $(Y', a_{i+1} \cdots a_j, \epsilon) \overset{c_2}{\vdash^*} (Y, \epsilon, v_2)$, for some stack symbols X, Y', Y, some input positions i and j $(0 \leq i \leq j \leq n)$, and some output string v_2. We express the existence of such a subcomputation by a different kind of table item, written as $p_2 : \mathit{inner}(X, Y, i, j)$, where $p_2 = p_{\mathcal{A}}(\tau c_2)$. Here, p_2 is related to what is commonly called an *inner* probability, as it expresses only the probability internally in a subcomputation.[7]

[6] Forward probability as defined by [49] refers to the sum of the probabilities of *all* computations from the beginning onward that lead to a certain rule occurrence, whereas here we consider only one computation at a time. We will turn to forward probabilities later in this section.

[7] We will turn to actual inner probabilities later in this section.

For technical reasons, we also need to consider computations c where $(X_{init}, a_1 \cdots a_j, \epsilon) \overset{c}{\vdash^*} (Y, \epsilon, v)$, for some Y, j and v. These are represented by table items $p_1 : \mathit{forward}(\perp, Y, 0, j)$, where $p_1 = p_{\mathcal{A}}(c)$. The symbol $\perp$ can be seen as an imaginary stack symbol that is located below the actual bottom-of-stack element.

All table items of the above forms, and only those table items, can be derived by the deduction system in Figure 5. Deduction systems for defining parsing algorithms have been described before by [43]; see also [44, 45] for a very similar framework. A dynamic programming algorithm for such a deduction system incrementally fills a *parse table* with table items, given a grammar and input. During execution of the algorithm, items that are already in the table are matched against antecents of inference rules. If a combination of items match all antecents of an inference rule, then the item that matches the consequent of that inference rule is added to the table. This process ends when no more new items can be added to the table.

The item in the consequent of inference rule (20) represents the fact that at the beginning of any computation, X_{init} lies on top of imaginary stack element $\perp$, no input has as yet been read, and the product of probabilities of all transitions used in the represented computation is 1, since no transitions have been used yet.

Inference rule (21) derives a table item from an existing table item, if the second stack symbol of that existing item indicates that a push transition can be applied. Naturally, the probability in the new item is the product of the probability in the old item and the probability of the applied transition. Inference rule (22) is very similar.

Two subcomputations are combined through a pop transition by inference rule (23), the intuition of which can be explained as follows. If W occurs as top-of-stack at position i and reading the input up to j results in Y on top of W, and if subsequently reading the input from j to k results in X on top of Y and YX may be replaced by Z by a pop transition, then reading the input from i to k results in Z on top of W. The probability of the newly derived subcomputation is the product of three probabilities. The first is the probability of that subcomputation up to the point where Y is top-of-stack, which is given by p_1; the second is the probability from this point onward, up to the point where X is top-of-stack, which is given by p_2; the third is the probability of the pop transition. The second of these probabilities, p_2, is defined by the inference rules for 'inner' items to be discussed next.

Inference rule (24) starts the investigation of a new subcomputation that begins with a push transition. This rule does not have any antecedents, but we may add an item $p_1 : \mathit{forward}(Z, X, i, j)$ as antecedent, since the resulting 'inner' items can only be useful for the computation of 'forward' items if at least one item of the form $p_1 : \mathit{forward}(Z, X, i, j)$ exists. We will not do so however, since this would complicate the theoretical analysis.

The next two rules, (25) and (26), are almost identical to (22) and (23).

It is not difficult to see that for each complete computation of the form

Initialization:

$$\frac{}{1 : \mathit{forward}(\bot, X_{\mathit{init}}, 0, 0)} \tag{20}$$

Push (forward):

$$\frac{p_1 : \mathit{forward}(Z, X, i, j)}{p_1 \cdot p_{\mathcal{A}}(\tau) : \mathit{forward}(X, Y, j, j)} \left\{ \; \tau = X \mapsto XY \right. \tag{21}$$

Scan (forward):

$$\frac{p_1 : \mathit{forward}(Z, X, i, j)}{p_1 \cdot p_{\mathcal{A}}(\tau) : \mathit{forward}(Z, Y, i, j')} \left\{ \begin{array}{l} \tau = X \overset{x,y}{\mapsto} Y \\ (x = \epsilon \wedge j' = j) \vee \\ \quad (x = a_{j+1} \wedge j' = j + 1) \end{array} \right. \tag{22}$$

Pop (forward):

$$\frac{\begin{array}{c} p_1 : \mathit{forward}(W, Y, i, j) \\ p_2 : \mathit{inner}(Y, X, j, k) \end{array}}{p_1 \cdot p_2 \cdot p_{\mathcal{A}}(\tau) : \mathit{forward}(W, Z, i, k)} \left\{ \; \tau = \; YX \mapsto Z \right. \tag{23}$$

Push (inner):

$$\frac{}{p_{\mathcal{A}}(\tau) : \mathit{inner}(X, Y, j, j)} \left\{ \; \tau = X \mapsto XY \right. \tag{24}$$

Scan (inner):

$$\frac{p_2 : \mathit{inner}(Z, X, i, j)}{p_2 \cdot p_{\mathcal{A}}(\tau) : \mathit{inner}(Z, Y, i, j')} \left\{ \begin{array}{l} \tau = X \overset{x,y}{\mapsto} Y \\ (x = \epsilon \wedge j' = j) \vee \\ \quad (x = a_{j+1} \wedge j' = j + 1) \end{array} \right. \tag{25}$$

Pop (inner):

$$\frac{\begin{array}{c} p_2 : \mathit{inner}(W, Y, i, j) \\ p_2' : \mathit{inner}(Y, X, j, k) \end{array}}{p_2 \cdot p_2' \cdot p_{\mathcal{A}}(\tau) : \mathit{inner}(W, Z, i, k)} \left\{ \; \tau = \; YX \mapsto Z \right. \tag{26}$$

Figure 5: Deduction system of table items.

$(X_{\mathit{init}}, a_1 \cdots a_n, \epsilon) \overset{c}{\vdash^*} (X_{\mathit{final}}, \epsilon, v)$, for some output string v, there is precisely one derivation by the deduction system of some table item $p_1 : \mathit{forward}(\bot, X_{\mathit{final}}, 0, n)$, where $p_1 = p_{\mathcal{A}}(c)$. Conversely, for each derivation of such a table item, there is a unique corresponding computation. Computations and derivations can be easily related to each other by looking at the transitions in the side conditions of the inference rules.

If follows that if we take the sum of p_1 over all derivations of items $p_1 : \mathit{forward}(\bot, X_{\mathit{final}}, 0, n)$, then we obtain the probability assigned by $\mathcal{A}$ to the input $w = a_1 \cdots a_n$.

Now assume that $\mathcal{A}$ is proper and consistent. For a given string $w' \in \Sigma_1^*$, where Σ_1 is the input alphabet, we define the *prefix probability* of w' to be

$$\sum_{w'' \in \Sigma_1^*} p_{\mathcal{A}}(w'w'')$$

In other words, we sum the probabilities of all strings $w = w'w''$ that start with prefix w'. We will now show that this probability can also be expressed in terms of the probabilities of 'forward' items.

Assume that $w' = a_1 \cdots a_n$, for some $n \geq 0$. Any computation on a string $w = w'w''$ that is the prefix of a complete computation must be of one of two types. The first is $(X_{init}, a_1 \cdots a_n, \epsilon) \overset{c}{\vdash^*} (X_{\mathit{final}}, \epsilon, v)$, for some v, which means that $w'' = \epsilon$, so that no input beyond position n needs to be read. The second is $(X_{init}, a_1 \cdots a_n a_{n+1} \cdots a_m, \epsilon) \overset{c_1}{\vdash^*} (\alpha X, a_{n+1} \cdots a_m, v_1) \overset{\tau}{\vdash} (\alpha Y, a_{n+2} \cdots a_m, v_1 y) \overset{c_2}{\vdash^*} (X_{\mathit{final}}, \epsilon, v_1 y v_2)$, where τ is a scan transition $X \overset{a,y}{\mapsto} Y$ such that $a = a_{n+1}$.

The sum of probabilities of computations of the first type equals the sum of p_1 over all derivations of items $p_1 : \mathit{forward}(\perp, X_{\mathit{final}}, 0, n)$, as we have explained above. For the second type of computation, properness and consistency implies that for given c_1 and τ as above, the sum of probabilities of different c_2 must be 1. (If that sum, say q, is less than 1, then the sum of the probabilities of all computations cannot be more than $1 - (1-q) \cdot p_{\mathcal{A}}(c_2) < 1$, which is in conflict with the assumed consistency.) Furthermore, properness implies that the sum of probabilities of different τ that we can apply for top-of-stack X must be 1. Therefore, we may conclude that the sum of probabilities of computations of the second type equals the sum of $p_{\mathcal{A}}(c_1)$ over all computations $(X_{init}, a_1 \cdots a_n, \epsilon) \overset{c_1}{\vdash^*} (\alpha X, \epsilon, v_1)$ such that there is at least one scan transition of the form $X \overset{a,y}{\mapsto} Y$. This equals the sum of p_1 over all derivations of items $p_1 : \mathit{forward}(Z, X, 0, n)$, for some Z, such that there is at least one scan transition of the form $X \overset{a,y}{\mapsto} Y$.

Hereby we have shown how both the probability and the prefix probability of a string can be expressed in terms of derivations of table items. However, the number of derivations of table items can be infinite. The obvious remedy lies in an alternative interpretation of the inference rules in Figure 5, following [16]: we regard objects of the form $\mathit{forward}(X, Y, i, j)$ or $\mathit{inner}(X, Y, i, j)$ as table items in their own right, and store each at most once in the parse table. The associated probabilities are then no longer those for individual derivations, but are the sums of probabilities over all derivations of table items $\mathit{forward}(X, Y, i, j)$ or $\mathit{inner}(X, Y, i, j)$. Such a sum of probabilities over all derivations of a table item is commonly called a *forward* or *inner* probability, respectively.

We will make this more concrete, under the assumption that there are no cyclic dependencies, i.e., there is no item $\mathit{forward}(X, Y, i, j)$ or $\mathit{inner}(X, Y, i, j)$ that may occur as ancestor of itself in some derivation. Let T be the set of all items $\mathit{forward}(X, Y, i, j)$ or $\mathit{inner}(X, Y, i, j)$ that can be derived using the deduction system in Figure 5, ignoring the probabilities. We then define a function p_{tab} from table items to probabilities, as shown in Figure 6. We assume the function

$$
\begin{aligned}
&p_{tab}(\mathit{forward}(X,Y,i,j)) \;=\; \qquad (27)\\
&\quad \delta(X=\bot \wedge Y = X_{init} \wedge i=j=0) \;+\\
&\quad \delta(i=j)\cdot \sum_{\substack{Z,k,\tau:\\ \mathit{forward}(Z,X,k,i)\in T,\\ \tau = X\mapsto XY}} p_{tab}(\mathit{forward}(Z,X,k,i))\cdot p_{\mathcal{A}}(\tau) \;+\\
&\quad \sum_{\substack{Z,j',x,y,\tau:\\ \mathit{forward}(X,Z,i,j')\in T,\\ (x=\epsilon\wedge j'=j)\vee(x=a_j\wedge j'=j-1),\\ \tau = Z\overset{x,y}{\mapsto} Y}} p_{tab}(\mathit{forward}(X,Z,i,j'))\cdot p_{\mathcal{A}}(\tau) \;+\\
&\quad \sum_{\substack{W,Z,k,\tau:\\ \mathit{forward}(X,W,i,k)\in T, \mathit{inner}(W,Z,k,j)\in T,\\ \tau = WZ\mapsto Y}} p_{tab}(\mathit{forward}(X,W,i,k))\cdot p_{tab}(\mathit{inner}(W,Z,k,j))\cdot p_{\mathcal{A}}(\tau)
\end{aligned}
$$

$$
\begin{aligned}
&p_{tab}(\mathit{inner}(X,Y,i,j)) \;=\; \qquad (28)\\
&\quad \delta(i=j)\cdot \sum_{\substack{\tau:\\ \tau = X\mapsto XY}} p_{\mathcal{A}}(\tau) \;+\\
&\quad \sum_{\substack{Z,j',x,y,\tau:\\ \mathit{inner}(X,Z,i,j')\in T,\\ (x=\epsilon\wedge j'=j)\vee(x=a_j\wedge j'=j-1),\\ \tau = Z\overset{x,y}{\mapsto} Y}} p_{tab}(\mathit{inner}(X,Z,i,j'))\cdot p_{\mathcal{A}}(\tau) \;+\\
&\quad \sum_{\substack{W,Z,k,\tau:\\ \mathit{inner}(X,W,i,k)\in T, \mathit{inner}(W,Z,k,j)\in T,\\ \tau = WZ\mapsto Y}} p_{tab}(\mathit{inner}(X,W,i,k))\cdot p_{tab}(\mathit{inner}(W,Z,k,j))\cdot p_{\mathcal{A}}(\tau)
\end{aligned}
$$

Figure 6: Recursive functions to determine probabilities of table items.

δ evaluates to 1 if its argument is true, and to 0 otherwise.

Each line in the right-hand sides of the two equations in Figure 6 can be seen as the backward application of an inference rule from Figure 5. In other words, for a given item, we investigate all possible ways of deriving that item as the consequent of different inference rules with different antecedents. For example, the second line in the right-hand side of equation (27), can be seen as the backward application of inference rule (21).

That Figure 6 is indeed equivalent to Figure 5 follows from the fact that multiplication distributes over addition. If there are cyclic dependencies, then the set of equations in Figure 6 may no longer have a closed-form solution, but we may obtain probabilities by an iterative algorithm that approximates the lowest non-negative solution to the equations [49].

Given the set of equations in Figure 6 we can now express the probability of a string of length n as $p_{tab}(\mathit{forward}(\bot, X_{\mathit{final}}, 0, n))$. The prefix probability of a string of length n is given by:

$$p_{tab}(\mathit{forward}(\perp, X_{\mathit{final}}, 0, n)) \ + \tag{29}$$

$$\sum_{\substack{X,Y,i: \\ \mathit{forward}(X,Y,i,n) \in T, \\ \exists \tau,a,y,Z[\tau = Y \overset{a,y}{\mapsto} Z]}} p_{tab}(\mathit{forward}(X, Y, i, n)) \tag{30}$$

To obtain a suitable PPDT from a given PCFG, we may apply the strategy $\mathcal{S}_{\epsilon\text{-}LC}$ from Section 6. Provided the (P)CFG is acyclic, this strategy ensures that there are no computations of infinite length for any given input, which implies there are no cyclic dependencies in the simulation of the automaton by the dynamic programming algorithm.

Hereby we have presented a way to compute probabilities and prefix probabilities of strings. Our approach is an alternative to the one from [36, 20, 49], and has the advantage that the approach is parameterized by the parsing strategy: instead of $\mathcal{S}_{\epsilon\text{-}LC}$ we may apply any other parsing strategy with the same properties with regard to acyclic grammars. If our grammars are even more constrained, e.g. if they do not have epsilon rules, we may apply even simpler parsing strategies. Different parsing strategies may differ in the efficiency of the computation.

10 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 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. Lastly, we have presented an application to prefix probabilities.

Acknowledgements

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

References

[1] S. Abney, D. McAllester, and F. Pereira. Relating probabilistic grammars and automata. In *37th Annual Meeting of the Association for Computational*

Linguistics, Proceedings of the Conference, pages 542–549, Maryland, USA, June 1999.

[2] S. Billot and B. Lang. 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, June 1989.

[3] N. Blum and R. Koch. Greibach normal form transformation revisited. *Information and Computation*, 150:112–118, 1999.

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

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

[6] E. Charniak. *Statistical Language Learning.* MIT Press, 1993.

[7] E. Charniak. Immediate-head parsing for language models. In *39th Annual Meeting and 10th Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 116–123, Toulouse, France, July 2001.

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

[9] C. Chelba and F. Jelinek. Exploiting syntactic structure for language modeling. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, volume 1, pages 225–231, Montreal, Quebec, Canada, August 1998.

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

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

[12] M.V. Chitrao and R. Grishman. Statistical parsing of messages. In *Speech and Natural Language, Proceedings*, pages 263–266, Hidden Valley, Pennsylvania, June 1990.

[13] M. Collins. Three generative, lexicalised models for statistical parsing. In *35th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 16–23, Madrid, Spain, July 1997.

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

[15] J. Goldstine, K. Price, and D. Wotschke. A pushdown automaton or a context-free grammar — which is more economical? *Theoretical Computer Science*, 18:33–40, 1982.

[16] J. Goodman. Semiring parsing. *Computational Linguistics*, 25(4):573–605, 1999.

[17] M.A. Harrison. *Introduction to Formal Language Theory.* Addison-Wesley, 1978.

[18] T. Huang and K.S. Fu. On stochastic context-free languages. *Information Sciences*, 3:201–224, 1971.

[19] K. Inui, V. Sornlertlamvanich, H. Tanaka, and T. Tokunaga. Probabilistic GLR parsing. In H. Bunt and A. Nijholt, editors, *Advances in Probabilistic and other Parsing Technologies*, chapter 5, pages 85–104. Kluwer Academic Publishers, 2000.

[20] F. Jelinek and J.D. Lafferty. Computation of the probability of initial substring generation by stochastic context-free grammars. *Computational Linguistics*, 17(3):315–323, 1991.

[21] D. Jurafsky and J.H. Martin. *Speech and Language Processing.* Prentice-Hall, 2000.

[22] D.E. Knuth. On the translation of languages from left to right. *Information and Control*, 8:607–639, 1965.

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

[24] R. Leermakers. 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, June 1989.

[25] H. Leung and D. Wotschke. On the size of parsers and LR(k)-grammars. *Theoretical Computer Science*, 242:59–69, 2000.

[26] C.D. Manning and B. Carpenter. Probabilistic parsing using left corner language models. In H. Bunt and A. Nijholt, editors, *Advances in Probabilistic and other Parsing Technologies*, chapter 6, pages 105–124. Kluwer Academic Publishers, 2000.

[27] C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing.* Massachusetts Institute of Technology, 1999.

[28] M.-J. Nederhof. Generalized left-corner parsing. In *Sixth Conference of the European Chapter of the Association for Computational Linguistics, Proceedings of the Conference*, pages 305–314, Utrecht, The Netherlands, April 1993.

[29] M.-J. Nederhof. 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, June 1994.

[30] M.-J. Nederhof and J.J. Sarbo. Increasing the applicability of LR parsing. In H. Bunt and M. Tomita, editors, *Recent Advances in Parsing Technology*, chapter 3, pages 35–57. Kluwer Academic Publishers, 1996.

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

[32] M.-J. Nederhof and G. Satta. An alternative method of training probabilistic LR parsers. In *42nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 551–558, Barcelona, Spain, July 2004.

[33] M.-J. Nederhof and G. Satta. Probabilistic parsing strategies. In *42nd Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference*, pages 543–550, Barcelona, Spain, July 2004.

[34] S.-K. Ng and M. Tomita. Probabilistic LR parsing for general context-free grammars. In *Proc. of the Second International Workshop on Parsing Technologies*, pages 154–163, Cancun, Mexico, February 1991.

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

[36] E. Persoon and K.S. Fu. Sequential classification of strings generated by SCFG's. *International Journal of Computer and Information Sciences*, 4(3):205–217, 1975.

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

[38] B. Roark and M. Johnson. 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, June 1999.

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

[40] J.-A. Sánchez and J.-M. Benedí. Consistency of stochastic context-free grammars from probabilistic estimation based on growth transformations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(9):1052–1055, September 1997.

[41] E.S. Santos. Probabilistic grammars and automata. *Information and Control*, 21:27–47, 1972.

[42] E.S. Santos. Probabilistic pushdown automata. *Journal of Cybernetics*, 6:173–187, 1976.

[43] S.M. Shieber, Y. Schabes, and F.C.N. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24:3–36, 1995.

[44] K. Sikkel. *Parsing Schemata*. Springer-Verlag, 1997.

[45] K. Sikkel and A. Nijholt. Parsing of context-free languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages. Vol 2: Linear Modeling: Background and Applications*, chapter 2, pages 61–100. Springer-Verlag, 1997.

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

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

[48] V. Sornlertlamvanich, K. Inui, H. Tanaka, T. Tokunaga, and T. Takezawa. Empirical support for new probabilistic generalized LR parsing. *Journal of Natural Language Processing*, 6(3):3–22, 1999.

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

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

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

[52] D.H. Van Uytsel and D. Van Compernolle. Language modeling with probabilistic left corner parsing. *Computer Speech and Language*, 19:171–204, 2005.

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

[54] F. Voisin. A bottom-up adaptation of Earley's parsing algorithm. In *Programming Languages Implementation and Logic Programming, International Workshop*, Lecture Notes in Computer Science, volume 348, pages 146–160, Orléans, France, May 1988. Springer-Verlag.

[55] J. Wright, A. Wrigley, and R. Sharman. Adaptive probabilistic generalized LR parsing. In *Proc. of the Second International Workshop on Parsing Technologies*, pages 100–109, Cancun, Mexico, February 1991.

[56] J.H. Wright and E.N. Wrigley. GLR parsing with probability. In M. Tomita, editor, *Generalized LR Parsing*, chapter 8, pages 113–128. Kluwer Academic Publishers, 1991.