STOCHASTIC PARSE-TREE RECOGNITION BY A PUSHDOWN AUTOMATON

Frédéric TENDEAU
INRIA-Rocquencourt [1], BP 105, 78153 Le Chesnay CEDEX, FRANCE
E-mail : Frederic.Tendeau@inria.fr

Abstract

We present the stochastic generalization of what is usually called correctness theorems : we guarantee that the probabilities computed operationally by the parsing algorithms are the same as those defined denotationally on the trees and forests defined by the grammar.

The main idea of the paper is to precisely relate the parsing strategy with a parse-tree exploration strategy : a computational path of a parsing algorithm simply performs an exploration of a parse-tree for the input portion already parsed. This approach is applied in particular to Earley and Left-Corner parsing algorithms. Probability computations follow parsing operations : looping problems (in rule prediction and subtree recognition) are solved by introducing probability variables (which may not be immediately evaluated). Convergence is ensured by the syntactic construction that leads to stochastic equations systems, which are solved as soon as possible.

Our algorithms accept any (probabilistic) CF grammar. No restrictions are made such as prescribing normal form, proscribing empty rules or cyclic grammars.

Keywords : pushdown automaton, dynamic programming, stochastic context-free grammar.

1 Introduction.

Stochastic context-free grammars are extensively used in natural language parsing and have been recently applied to genome analysis. They define probabilities on rules, hence on derivations and finally on the language described. These probabilities may be computed to rank the different derivations of an ambiguous sentence or to estimate the probability of sentences with a given prefix. In general, the relevant literature does not give the formal framework that could permit to guarantee the correctness of computations. [Booth-Thompson 73] gives such a framework, to study the problem of adequacy between probabilities in the grammar and in the language.

Our aim is to present a Left Corner (LC) algorithm that recognizes stochastic parse forests and computes prefix and subtree probabilities, improving [Stolcke 93] which is based on Earley's algorithm. We also attempt to address more systematically the issue of the correctness of the stochastic parser w.r.t. the denotational definition of sentence probabilities by the grammar.

After the presentation of the formalism, we progress towards our goal step by step, developing each time one of the following directions :

parsing strategy : before the LC strategy, we describe Earley's, which is in fact the same strategy without grammar compilation. The parsing strategies are expressed in term of pushdown automata (PDA).

decoration level : before the probabilistic computations, we give a purely syntactic version.

number of considered trees : before considering shared forests, we present the problem on a single tree. The shared forest is produced by a dynamic programming interpretation of the PDA and is expressed in term of a pushdown transducer (PDT).

Analyzing the various combinations may be viewed as exploring the eight corners of a cube. This separation of issues allows a uniform presentation of all variants. It also makes the algorithms easier to understand, and to prove correct.

[1] this work was partially supported by the grant 95-B030 from the Centre National d'Études des Télécommunications.

2 Analysis Material.

2.1 Grammars.

Definition 1. A *context-free grammar* (CFG) is a 4-tuple $(\Sigma, \mathcal{N}, \mathcal{R}, S)$, where Σ is the set of terminal symbols, $\mathcal{N}$ that of non-terminals ($\Sigma \cap \mathcal{N} = \emptyset$), $\mathcal{R}$ that of rules, and $S \in \mathcal{R}$ is the axiom. The set $\mathcal{V} = \Sigma \cup \mathcal{N}$ is called the vocabulary.

We use usual notations: early Latin lowercase ($a, b...$) for terminals, late ones (x, y) for terminal strings, early Latin uppercase alphabet (A,B...) for the non-terminals, late ones (X,Y) for vocabulary symbols, and early Greek letters ($\alpha, \beta...$) for the vocabulary strings.

The grammar rules are also called productions. They are noted $A \rightarrow \alpha$, with $A \in \mathcal{N}$ called the left hand side (LHS) and $\alpha \in \mathcal{V}^*$ called right hand side (RHS). A rule having A as LHS is an A-rule. The collection of CFGs is noted $\mathcal{G}$.

From now on we implicitly consider a CFG $G = (\Sigma, \mathcal{N}, \mathcal{R}, S)$.

2.2 Derivation and Language.

Usual definitions.

– The relation *derive* is $\Longrightarrow = \{(\alpha A \beta, \alpha\gamma\beta) \mid A \in \mathcal{N}, \alpha, \beta, \gamma \in \mathcal{V}^* \text{and } A \rightarrow \gamma \in \mathcal{R}\}$

The reflexive transitive closure is noted $\overset{*}{\Longrightarrow}$. Deriving n times is noted $\overset{n}{\Longrightarrow}$.

– A *derivation* is a sequence $(\alpha_i)_{i \geq 0}$ of $\mathcal{V}^*$ s.t. for each i, $\alpha_i \Longrightarrow \alpha_{i+1}$.

By definition, for each $0 \leq i \leq j$, $\alpha_i \overset{*}{\Longrightarrow} \alpha_j$: each sub-sequence is a derivation. We say that $(\alpha_k)_{k \in [i,j]}$ is a derivation of α_i into α_j. A derivation is *elementary* if it is a pair of which first element is a non-terminal: it is noted like a grammar rule.

– A non-terminal symbol A is *productive* iff exists a terminal string x s.t. $A \overset{*}{\Longrightarrow} x$.

– A non-terminal symbol A is *accessible* iff $S \overset{*}{\Longrightarrow} \alpha A \beta$.

– A vocabulary string $\alpha \in \mathcal{V}^*$ is a *sentential form* iff $S \overset{*}{\Longrightarrow} \alpha$.

– A terminal chain $x \in \Sigma^*$ is a *sentence* iff $S \overset{*}{\Longrightarrow} x$. (a sentence is a sentential form)

– The *language defined by G* is $\mathcal{L}(G) = \{x \in \Sigma^* \mid S \overset{*}{\Longrightarrow} x\}$.

– The relation *leftmost derive* is $\underset{\ell}{\Longrightarrow} = \{(xA\omega, x\eta\omega) \mid xA\omega \Longrightarrow x\eta\omega\}$

Leftmost derivations (sequences) and *leftmost sentential forms* are naturally derived. The set of all leftmost derivations of α into β is noted $\alpha \underset{\ell}{\rightsquigarrow} \beta$, and $\alpha \overset{n}{\underset{\ell}{\rightsquigarrow}} \beta = \{d \in \alpha \underset{\ell}{\rightsquigarrow} \beta \mid \text{length of } d \text{ is } n\}$.

– A *sentential form* α is *ambiguous* iff $S \underset{\ell}{\rightsquigarrow} \alpha$ contains at least two elements.

– A *grammar* G is *ambiguous* iff an ambiguous sentence can be derived from it.

Proposition 2. *Classical results.*

(*i*) If $A \overset{*}{\Longrightarrow} x$ then $A \overset{*}{\underset{\ell}{\Longrightarrow}} x$.

(*ii*) $\mathcal{L}(G) = \{x \in \Sigma^* \mid S \overset{*}{\underset{\ell}{\Longrightarrow}} x\}$.

Definition 3. The *set of leftmost derivations* is

$$\Delta_\ell = \{\delta_0, \delta_1\} \cup \bigcup_{\alpha, \beta \in \mathcal{V}^*} \alpha \underset{\ell}{\rightsquigarrow} \beta$$

where δ_1 (the unit derivation) and δ_0 (the inconsistent derivation) are defined w.r.t. the composition properties.

Definition 4. The *composition of leftmost derivations* is the function $\textcircled{\ell}$: $\Delta_\ell \times \Delta_\ell \longrightarrow \Delta_\ell$ s.t.

$\forall\, d \in \Delta_\ell$, $d \,\textcircled{\ell}\, \delta_1 = \delta_1 \,\textcircled{\ell}\, d = d$ and $d \,\textcircled{\ell}\, \delta_0 = \delta_0 \,\textcircled{\ell}\, d = \delta_0$.

$\forall\, (d)_{i \in [0,l]} \in \alpha \underset{\ell}{\rightsquigarrow} x\beta\gamma$ and $(d')_{i \in [0,l']} \in \beta \underset{\ell}{\rightsquigarrow} \nu$, $(d'')_{i \in [0,l'']} = d \,\textcircled{\ell}\, d' \in \alpha \underset{\ell}{\rightsquigarrow} x\nu\gamma$ is s.t. $l'' = l + l'$, $(d'')_{i \in [0,l]} = (d)_{i \in [0,l]}$, and for $i > l$, $d''_i = x d'_{i-l} \gamma$.

2.3 Trees and Forests.

We use the approach of [Lang 89] to define trees as grammars: $(\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ is a derivation tree from $(\Sigma, \mathcal{N}, \mathcal{R}, S)$.

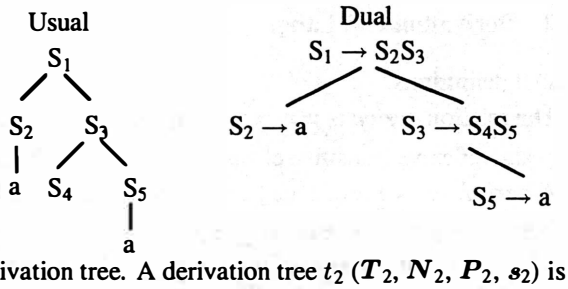
Definition 5. A *derivation tree* is the inconsistent tree τ_0, or the unit tree τ_1 or a CFG $(\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ verifying

1. for each $A \in \boldsymbol{N}$, A is accessible ; A is the LHS of at most one rule ; A appears at most once in at most one RHS.
2. $\boldsymbol{s}$ occurs in no RHS.
3. there exists a labelling function *lab*, from $(\boldsymbol{T} \cup \boldsymbol{N})^* \cup \boldsymbol{P}$ to $\mathcal{V}^* \cup \mathcal{R}$ s.t. $lab(\boldsymbol{N}) \subset \mathcal{N}$, $lab(\boldsymbol{P}) \subset \mathcal{R}$, $lab(a) = a$, $lab(\epsilon) = \epsilon$, $lab(X\alpha) = lab(X)lab(\alpha)$, $lab(A \rightarrow \alpha) = lab(A) \rightarrow lab(\alpha)$.

The set of derivation trees is $\mathcal{T}$. The vocabulary $\boldsymbol{V} = \boldsymbol{T} \cup \boldsymbol{N}$. Productions are called nodes, the $\boldsymbol{s}$-rule is called the root.

Example 6.

Given $\mathcal{R} = \{S \rightarrow SS, S \rightarrow a\}$, a possible derivation tree is s.t. $\boldsymbol{P} = \{S_1 \rightarrow S_2S_3, S_3 \rightarrow S_4S_5, S_2 \rightarrow a, S_5 \rightarrow a\}$ and $\forall i\ lab(S_i) = S$. This tree is shown as a dual form of the usual view.



Definition 7. Let t_1 $(\boldsymbol{T}_1, \boldsymbol{N}_1, \boldsymbol{P}_1, \boldsymbol{s}_1)$ be a derivation tree. A derivation tree t_2 $(\boldsymbol{T}_2, \boldsymbol{N}_2, \boldsymbol{P}_2, \boldsymbol{s}_2)$ is a *derivation subtree* of t_1 iff $\boldsymbol{P}_2 \subset \boldsymbol{P}_1$.

Definition 8. A parse tree t is a *derivation tree* $(\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ of which non-terminals are all productive, such that $lab(\boldsymbol{s}) = S$.

t is a parse tree of x iff $\mathcal{L}(t) = \{x\}$, alternately: t spans x.

During parsing, we need usual notion on trees (child, left and right) so we introduce orders :

– *parental or vertical order.* Given a derivation tree t, $\succeq \subset \boldsymbol{N} \times \boldsymbol{N}$ is the reflexive transitive closure of $\succ$: $A \succ B$ iff t has a rule $A \rightarrow \alpha B \beta$.

When $A \succeq B$, A is ancestor of B and B descendant of A — if $A \succ B$ then A is parent of B and B is a child of A.

– *horizontal order.* Let $t = (\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ be a tree, $\ll$ is a partial order defined on $\boldsymbol{N}$, and a relation on Σ: let X and Y be in $\boldsymbol{V}$, $X \ll Y$ iff $\boldsymbol{s} \stackrel{*}{\Longrightarrow} \alpha X \beta Y \gamma$.

"$X \ll Y$" is said "X *on the left of* Y".

Definition 9. The *composition of derivation trees* is the function $\otimes : \mathcal{T} \times \mathcal{T} \longrightarrow \mathcal{T}$ s.t.
$\forall\, t \in \mathcal{T}$, $t \otimes \tau_1 = \tau_1 \otimes t = t$ and $t \otimes \tau_0 = \tau_0 \otimes t = \tau_0$.
$\forall\, t_1 = (\boldsymbol{T}_1, \boldsymbol{N}_1, \boldsymbol{P}_1, \boldsymbol{s}_1)$ and $t_2 = (\boldsymbol{T}_2, \boldsymbol{N}_2, \boldsymbol{P}_2, \boldsymbol{s}_2)$, $t_1 \otimes t_2 = (\boldsymbol{T}_1 \cup \boldsymbol{T}_2, \boldsymbol{N}_1 \cup \boldsymbol{N}_2, \boldsymbol{P}_1 \cup \boldsymbol{P}_2, \boldsymbol{s}_1)$, if this grammar is a derivation tree and if $\boldsymbol{s}_2$ is the leftmost t_1 non-terminal without a child [2], else $t_1 \otimes t_2 = \tau_0$.

A derivation tree $(\boldsymbol{T}, \boldsymbol{N}, \boldsymbol{P}, \boldsymbol{s})$ is elementary iff $\boldsymbol{P}$ is a singleton. Derivations trees $\otimes$-composed with only elementary trees are called elementarily $\otimes$-composed. There exists a correspondence between leftmost derivations and elementary $\otimes$-composed derivation trees. From derivation trees to leftmost derivation, one has just to replace $\otimes$ by $\textcircled{\ell}$, and the nodes by their labels. From derivations to trees, $\textcircled{\ell}$ is replaced by $\otimes$ and a labelling function must be given.

2.3.1 Ambiguity.

Definition 10. A *parse forest* is a set of parse trees.

Let x be in $\mathcal{L}(G)$. The set of all the parse trees of x is called the parse forest of x. The composition $\textcircled{\ell}$ is naturally extended on Δ_ℓ subsets.

[2] this last condition is not necessary, *i.e.* $\otimes$ corresponds to the derivation, but we want $\otimes$ corresponds to the *leftmost* derivation.

2.4 Probabilistic Grammars and Forests.

Probabilities are defined on a set of events. What we call an event is the derivation of a non-terminal A using some grammar rule: an event requires a non-terminal A, we speak of an A-event. The set of all possible A-events is the set of ways to derive A, *i.e.* the set of A-rules. A probability is assigned to each rule: the probabilities over $\mathcal{R}$ are well defined iff the sum of all A-rules probability is 1. The probability domain is $\mathcal{D}(P)$.

Definition 11. A *probabilistic, or stochastic, CFG* is a 5-tuple $(\Sigma, \mathcal{N}, \mathcal{R}, S, P)$, s.t. $(\Sigma, \mathcal{N}, \mathcal{R}, S)$ is a CFG in which all non terminals are accessible, and P is a function: $\mathcal{R} \longrightarrow \mathcal{D}(P)$, that associates with each production $A \rightarrow \omega$ a probability $P(A \rightarrow \omega)$, and s.t. for each $A \in \mathcal{N}$, $\sum_{A \rightarrow \alpha \in \mathcal{R}} P(A \rightarrow \alpha) = 1$.

P is extended over derivations. A leftmost derivation is either δ_0, or δ_1, or an elementary derivation, or a composition of derivations: $P(\delta_0) = 0$; $P(\delta_1) = 1$; $P(A \overset{1}{\underset{\ell}{\rightsquigarrow}} \alpha) = P(A \rightarrow \alpha)$; for $d_1, d_2 \in \Delta_\ell$ s.t. $d_1 \textcircled{\ell} d_2 \neq \delta_0$, $P(d_1 \textcircled{\ell} d_2) = P(d_1) \times P(d_2)$.

$P(d)$ is the probability that an arbitrary derivation d', differing from a prefix of d by the last element of d', or equal to d, is actually d.

The equivalence between elementarily $\otimes$-composed derivation trees and leftmost derivations gives: for $t = (T, N, P, s)$, a derivation tree, $P(t) = \prod_{r \in P} P(lab(r))$. This is operationally used in our stochastic computations.

P is extended on sets of derivations. For $E \subset \Delta_\ell$, $P(E) = \sum_{d \in E} P(d)$.

Application to leftmost sentential forms: for α s.t. $S \overset{*}{\underset{\ell}{\Longrightarrow}} \alpha$, $P(\alpha) = P(S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha)$.

Definition 12. *Consistency condition.* A probabilistic grammar G is consistent iff

$$\lim_{n \to \infty} \sum_{x A \beta \text{ s.t. } S \overset{n}{\underset{\ell}{\Longrightarrow}} x A \beta} P(x A \beta) = 0$$

Proposition 13. If G is consistent then $P(\mathcal{L}(G)) = 1$.

The proof is in [Booth-Thompson 73], the idea is : starting from all the S-rules (probability = 1) and, for each, making all possible choices to derive the leftmost non-terminal (say A) at each step (choosing all the A-rules) keeps the sum of probabilities of all leftmost sentential forms equal to 1. When n grows to infinity there are more and more sentences but $P(\mathcal{L}(G)) = 1$ iff the sentential forms all tend to be sentences, *i.e.* the probability of the set of sentential forms that still contain a non-terminal tends to 0.

From now on our probabilistic grammars will be assumed consistent.

Corollary: this result states that the set of leftmost derivations from S to terminal strings can be considered as a set of independent events. The accessibility of each non terminal gives (i) for each $\alpha \in \mathcal{V}^*$, $\sum_x P(\alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x) = 1$. (ii) given a leftmost sentential form α, $P(S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha) = P(S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha) \sum_x P(\alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x)$ $= \sum_x P(S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha \textcircled{\ell} \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x)$ — the probability of the leftmost derivations from S to terminal strings, that contain α: $S \overset{*}{\underset{\ell}{\rightsquigarrow}} \alpha \textcircled{\ell} \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x \subset S \overset{*}{\underset{\ell}{\rightsquigarrow}} x$.

2.5 The Dotted Rules.

The dotted rules are used to describe the parsing process. We show they are a mean to describe positions in a parse-tree.

Definition 14. A *dotted rule* is a couple $(A \rightarrow \alpha, index)$, usually noted $A \rightarrow \omega \bullet \delta$ (with $|\omega| = index$), s.t. $0 \leq index \leq |\omega\delta|$.

index indicates how many symbols have been recognized in RHS.

$A \rightarrow \bullet\alpha$ and $A \rightarrow \alpha\bullet$ are respectively an initial and a final (or complete) dotted rule.

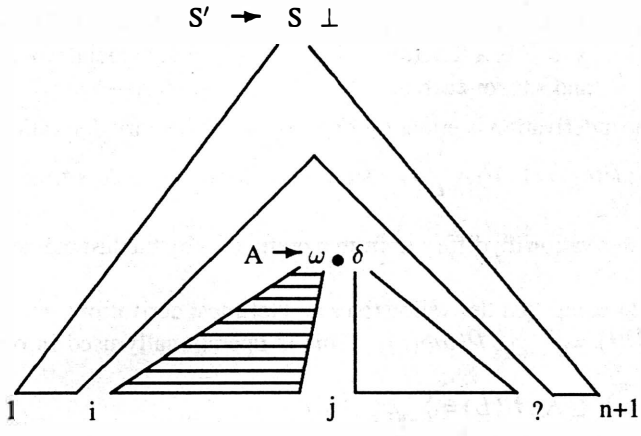
2.6 Parsing Conventions.

1. The grammar is augmented: a new symbol is added to $\mathcal{N}$: S' ; a new symbol is added to Σ: $\perp$; a new rule is added to $\mathcal{R}$: $S' \rightarrow S\perp$, s.t. $P(S' \rightarrow S\perp) = 1$.
2. For $w \in \Sigma^*$, $|w| = n$ and $1 \leq i \leq j \leq n$, $w_{i..j}$ stands for the string w index range $[i, j[$.

3. From now on we will consider an input string: $x_{1\cdots n+1}$, its length is n, its first token, x_1, and its last, x_n. The token x_{n+1} is $\perp$, which is not recognized by the non augmented grammar.

2.7 The Earley-Items.

Definition 15. An *Earley-item* is a triple noted $[A \rightarrow \omega \bullet \delta, i, j]$ s.t. $\omega \overset{*}{\Longrightarrow} x_{i\cdots j}$.



$[A \rightarrow \omega \bullet \delta, i, j]$ can be seen both as a sub-parse-tree and as a position on the node that dominates it: the subtree is the one that spans $x_{i\cdots j}$ (the one with horizontal lines), its root is labelled by $A \rightarrow \omega\delta$ and the dot • points the position between ω and δ. The indexes are between the leaves (the question mark states that the index of the position is unknown).

Initial and final dotted rules correspond to initial and final positions on a node.
The collection of Earley-items on G is $\mathcal{I}(G)$.

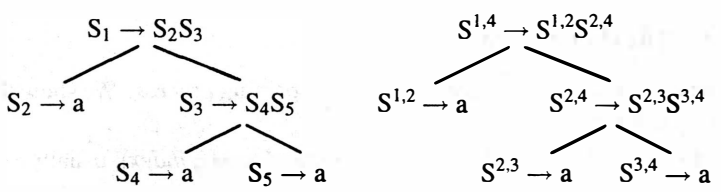
2.8 Shared Forest.

A function $\mathcal{G}$ will serve to define shared forests, using the Earley-items associated with each node of the parse trees. The set of natural integers is noted $\mathbb{N}$.

Definition 16. The *renaming parse-tree function* $\mathcal{G} : \mathcal{T} \longrightarrow \mathcal{G}$.
Consider a parse-tree $t = (T, N, P, s)$, and the sets $\mathcal{M} = \mathcal{N} \times \mathbb{N} \times \mathbb{N}$ and $\mathcal{W} = \mathcal{M} \cup \Sigma$. [3]
$\mathcal{G}(t) = (\Sigma, \mathcal{N}', \mathcal{R}', S_o)$, with $\mathcal{N}' \subset \mathcal{M}$ and $\mathcal{R}' \subset \mathcal{M} \times \mathcal{W}^*$.
$\mathcal{R}'$: let r be in P s.t. $lab(r) = A \rightarrow X_1, \ldots, X_k \in \mathcal{R}$. Let us consider the sequence of Earley-items on this node: $([A \rightarrow X_1 \ldots X_m \bullet X_{m+1} \ldots X_k, i, j_m])_{m \in [0,k]}$, (recall that $j_0 = i$).
Then $A^{i,j_k} \rightarrow Y_1, \ldots, Y_k \in \mathcal{R}'$ is built s.t. if $X_m \in \Sigma$ then $Y_m = X_m$, else $Y_m = X_m{}^{j_{m-1}, j_m}$.
$\mathcal{N}'$ elements are inferred from $\mathcal{R}'$.

A renamed parse-tree is no longer a parse-tree because a non-terminal can appear more than once in RHS, due to $A \overset{+}{\Longrightarrow} A$ cycles: note that two parse trees differing only by such a cycle have the same $\mathcal{G}$ image. However, the other conditions are respected. *e.g.* G' spans x (by definition of Earley-items) and the labelling function is s.t. $lab(A^{i,j}) = A$.

Example 17.
Opposite, we show the derivation tree of example 6., completed with a new rule $(S_4 \rightarrow a)$ to make a parse tree t, and next, $\mathcal{G}(t)$.



Definition 18. *Union of grammars.* $\cup : \mathcal{G} \times \mathcal{G} \longrightarrow \mathcal{G}$
$(\Sigma, \mathcal{N}_1, \mathcal{R}_1, S_o) \cup (\Sigma, \mathcal{N}_2, \mathcal{R}_2, S_o) = (\Sigma, \mathcal{N}_1 \cup \mathcal{N}_2, \mathcal{R}_1 \cup \mathcal{R}_2, S_o)$.

Definition 19. The *shared forest* of a sentence $x \in \mathcal{L}(G)$ is $\bigcup_{t \text{ parse-tree of } x} \mathcal{G}(t)$.

The number of parse trees for x may be infinite. However the sets of non-terminals and rules in the infinite union are all finite because the number of Earley-items for x is finite. Therefore the shared forest is a grammar.

[3] $\mathcal{W}$ will be used while describing dynamic programming interpretations.

In general, this grammar is not a parse-tree because a non-terminal can be in different LHSs (due to ambiguity and causing subtree sharing) or can appear several times in RHS (causing context sharing, see [Lang 89]).

In the shared forest context, the Earley-items $[A \rightarrow \omega \bullet \delta, i, j]$ can be seen as the sub-(shared) forest that spans $x_{i..j}$ and has a root labelled by $A \rightarrow \omega\delta$. It corresponds to this position in all the parse-trees.

2.9 Probabilities and Earley-Item.

Definition 20. The 5-tuple $< A \rightarrow \alpha \bullet \beta, i, j, \mathbb{P}, \mathbb{R} >$ is a *probabilistic item* iff $I = [A \rightarrow \alpha \bullet \beta, i, j]$ is an Earley-item s.t. $p_r(I) = \mathbb{R}$ and $p_p(I) = \mathbb{P}$, where p_r and p_p are defined below.

2.9.1 Recognition Probability.

As a corollary of the grammar consistency (proposition 13.), one proves that p_r is the probability of all the sub-parse trees rooted in a node labelled by $A \rightarrow \alpha\beta$, s.t. α spans $x_{i..j}$.

We define $p_r([A \rightarrow \alpha \bullet \beta, i, j]) = p(A \rightarrow \alpha\beta \,@\, \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j})$

2.9.2 Prediction Probability.

As above, one proves that p_p is the probability of an Earley-item, seen as a set of leftmost derivation trees corresponding to some single sentential form.

$p_p([A \rightarrow \alpha \bullet \beta, i, j]) = p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega \,@\, A \rightarrow \alpha\beta \,@\, \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j})$

Property 21. $p_p([A \rightarrow \alpha \bullet \beta, i, j]) = p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega).\ \times p_r([A \rightarrow \alpha \bullet \beta, i, j])$

Proof: $p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega \,@\, A \rightarrow \alpha\beta \,@\, \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j}) = p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega) \times p(A \rightarrow \alpha\beta \,@\, \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j})$.

Property 22. $p_p([S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1]) = p_r([S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1]) = p(x_{1..n+1})$.

Proof: $p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..1} S') = p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} S') = p(\delta_1) = 1$

therefore, using the previous property, $p_p([S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1]) = p_r([S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1])$.

Now, $p(S' \overset{*}{\underset{\ell}{\rightsquigarrow}} S\bot) = 1$ and $p(S \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..n+1}) = p(x_{1..n+1})$ give the second equality.

3 Pushdown Automata and their Interpretation.

We distinguish a (non-deterministic) PDA, that describes the parsing strategy, from its dynamic programming interpretation, presented in a pushdown transducer formalism.

3.1 Pushdown Automaton.

Definition 23. A PDA is a 4-tuple $(\Sigma_\pi, \pi^i, \pi^f, T)$ s.t. Σ_π is the stack alphabet ; π^i is the initial stack ; π^f is the final stack ; T is the set of transitions on the stack.

Stacks are noted $[\text{head}|\text{tail}]$.

Transitions, so : top1, input $\mapsto$ top2, where

– top1 is the stack 1 or 2 top element(s) before the transition ;

– input is the current token of the input : noted a if it is scanned, (a) if it is only read and () if the transition is input independent.

– top2 is the stack 1 or 2 top element(s) after the transition.

Without considering the input, there are four kinds of transitions :

SWAP : $e_1, input \mapsto e'_1$. The top element is consulted and popped, another is pushed.

PUSH : $e_2, input \mapsto e_1, e_2$. The top element is consulted and another is pushed.

POP : $e_1, e_2, input \mapsto e'_2$. The top element and the one beneath are consulted and popped, another is pushed.

SWAP2 : $e_1, e_2, input \mapsto e'_1, e_2$. The top element and the one beneath are consulted, only the first is popped, another is pushed.

A complete computation ends with the final stack.

3.2 The Interpretations.

An interpretation is an execution model of the automaton which manages the non-determinism with dynamic programming technic. We use [VdlC 93] terminology and the so-called S^1 interpretation.

S^1 destroys the stack structure but stores the elements in a set called 1-items set : $\mathcal{E}$. The transitions re-build some parts of the stack.

For example the 1-items $[a]$ and $[b]$, permit to build 2 stacks : $p = [a, b]$ and $q = [b, a]$. But a POP requiring the top (a,b) considers only p.

S^1 interpretation is very compact, and allows very good sharing between the stacks. The form of the elements (Earley-items) will guarantee the correctness of the execution.

Definition 23. A *pushdown transducer* is a 5-tuple $(\Sigma_\pi, \pi^i, \pi^f, T, \Sigma_\sigma)$ s.t. $(\Sigma_\pi, \pi^i, \pi^f, T)$ is a PDA and Σ_σ is the output alphabet.

PDTs will be used for dynamic programming interpretations of PDA. The stack of our PDTs will be the set of 1-items $\mathcal{E}$. The transitions are the same than for PDA, excepted that they have an extra parameter for the output. They have the form : top1, input $\mapsto$ top2, output, with ε meaning no output. Stacks elements on the left (top1) are chosen in $\mathcal{E}$, and top2 is a single element, added to $\mathcal{E}$. This makes PUSH look like SWAP, and POP like SWAP2.

The execution starts with the top element of π^i in $\mathcal{E}$, it loops until no more 1-item can be added to $\mathcal{E}$ or no more token is to be read.

The execution output is the shared-parse-forest of the input.

4 Earley.

In [Earley 70] a CFG recognizer and parser are given. The recognizer uses dynamic programming, what leads to a cubic time and space complexity. The parsing strategy it uses is not separated from its dynamic programming interpretation. We do such a separation.

4.1 Earley's PDA.

From the augmented grammar $(\Sigma \cup \{\bot\}, \mathcal{N} \cup \{S'\}, \mathcal{R} \cup \{S' \rightarrow S\bot\}, S')$ and $x_{1..n+1}$, a PDA $(\Sigma_\pi, \pi^i, \pi^f, T)$ is built s.t.

Stack alphabet : $\Sigma_\pi = \mathcal{I}(G) \cup \{\bot\}$.

Initial stack : $\pi^i = \left[[S' \rightarrow \bullet S\bot, 1, 1] \middle| \bot\right]$

Final stack : $\pi^f = \left[[S' \rightarrow S^{1,n+1} \bullet \bot, 1, n+1] \middle| \bot\right]$

Transitions : T = PUSH $\cup$ SWAP $\cup$ POP.

PUSH : set of *prediction* transitions, of the form
$[A \rightarrow \alpha \bullet B\beta, i, j], () \mapsto [B \rightarrow \bullet\gamma, j, j], [A \rightarrow \alpha \bullet B\beta, i, j]$

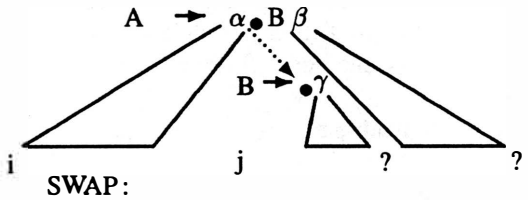
SWAP : set of *scan* transitions, of the form
$[A \rightarrow \alpha \bullet a\beta, i, j], a \mapsto [A \rightarrow \alpha a \bullet \beta, i, j+1]$

POP : set of *completion* transitions, of the form
$[A \rightarrow \alpha \bullet, i, j], [B \rightarrow \eta \bullet A\beta, k, i], () \mapsto [B \rightarrow \eta A \bullet \beta, k, j]$

4.2 Proof Indications.

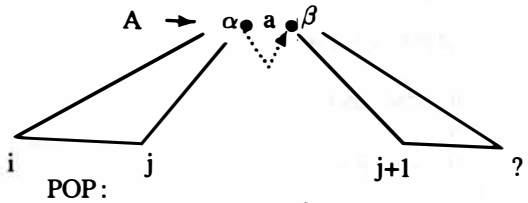
For correction: first, the initial Earley-item corresponds to top-leftmost position on any parse-tree. Afterwards, assuming we have a deterministic complete computation, the figures below show that each computation step consists in a left-to-right depth-first exploration (LRDFE) step of a $x_{1..n+1}$ parse-tree.

PUSH:



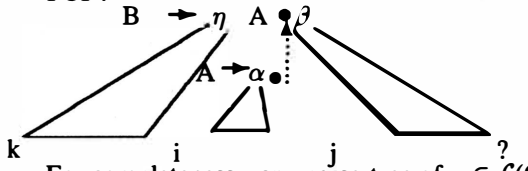
The current position is on a node labelled by $A \rightarrow \alpha B \beta$, after α and in front of B, the computation pushes the initial position on a node labeled by $B \rightarrow \gamma$. It corresponds to going down to the current node next child to the right: a LRDFE step.

SWAP:



Here, the computation corresponds to staying in the same node and moving to the next position from left to right, a token being recognized.

POP:



The current node N is labeled by $A \rightarrow \alpha$ and the position is final, a whole sub-parse tree has been recognized. The computation corresponds to the come back to its parent, on N right.

For completeness: any parse-tree of $x \in \mathcal{L}(G)$ can be explored LRDF. Each movement is either a left to right step on a node, or a descent to a child, or a come back up to a parent, to which corresponds respectively a SWAP, PUSH or POP transition, and only one.

Note that the extended version of this paper, with all the complete proofs, is to appear in [Tendeau 95].

4.3 Earley-Item and Left Derivation.

A LRDFE, with an action executed on each node the first time it is met, is a prefix LRDFE. Considering the output of the rule that labels a node as an action, a prefix LRDFE of a given parse-tree yields a left derivations of that tree.

We used to see Earley-items as positions in a parse-tree, the Earley parsing strategy permits to consider them also as a step within a (prefix) LRDFE, *i.e.* within a left derivation process: $I = [A \rightarrow \alpha \bullet \beta, i, j]$ identifies $S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega \; \textcircled{\ell} A \rightarrow \alpha\beta \; \textcircled{\ell} \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j}$. [4]

Hence $p_p(I)$ is the probability of the left derivation trees of $S' \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{1..i} A\omega \; \textcircled{\ell} A \rightarrow \alpha\beta \; \textcircled{\ell} \alpha \overset{*}{\underset{\ell}{\rightsquigarrow}} x_{i..j}$.

This permits to present the intuition of the proofs: a parsing algorithm performs a LRDFE and the probability of explored (sub-)trees are incrementally computed.

4.4 Probabilistic PDA.

Grammar rules are now decorated by probabilities. Trees and automata are also decorated.

Stack alphabet $\mathcal{I}(G) \times \mathcal{D}(p) \times \mathcal{D}(p) \cup \{\bot\}$.

Initial stack $[\langle S' \rightarrow \bullet S \bot, 1, 1, 1, 1 \rangle | \bot]$

Final stack $[\langle S' \rightarrow S \bullet \bot, 1, n+1, \Pi, \Pi \rangle | \bot]$

PUSH

$\langle A \rightarrow \alpha \bullet B\delta, i, j, \mathbf{P}, \mathbf{R} \rangle, () \mapsto \langle B \rightarrow \bullet\eta, j, j, \mathbf{P}p(B \rightarrow \eta), P(B \rightarrow \eta) \rangle, \langle A \rightarrow \alpha \bullet B\delta, i, j, \mathbf{P}, \mathbf{R} \rangle$

SWAP

$\langle A \rightarrow \alpha \bullet a\delta, i, j, \mathbf{P}, \mathbf{R} \rangle, a \mapsto \langle A \rightarrow \alpha a \bullet \delta, i, j+1, \mathbf{P}, \mathbf{R} \rangle$

[4] when considering a deterministic computation this set is reduced to a singleton.

POP

$$<A\rightarrow\alpha\bullet, i, j, \mathbf{P}'', \mathbf{R}''>, <B\rightarrow\eta\bullet A\beta, k, i, \mathbf{P}, \mathbf{R}>, () \mapsto <B\rightarrow\eta A\bullet\beta, k, j, \mathbf{PR}'', \mathbf{RR}''>$$

4.5 Proof indications.

SWAP : the position progresses but in the same node, hence the probabilities remain unchanged.

PUSH : a node N, that has never been visited, is reached. The prediction probability is multiplied by $P(lab(N))$. The subtree identified by the pushed item has only one node, N, then $P(lab(N))$ is its probability.

POP : I,J $\mapsto$ K. K identifies J derivation tree composed with I sub-parse tree. Then K probabilities are J ones multiplied by $P_p(I)$.

5 Dynamic Programming Interpretation of Earley's Automata.

To produce the shared forest, we rename (using g) the sub-parse trees incrementally. When a non-terminal $N \in \mathcal{N}$ is shifted in a completion, it is immediately renamed by $N' \in \mathcal{M}$: the following Earley-items $[A\rightarrow\alpha\bullet\beta, i, j]$ are s.t. $\alpha \in \mathcal{W}^*$, and $\beta \in \mathcal{V}^*$, their set is $\mathcal{I}(G)^I$.

5.1 Earley PDA Interpretation.

$\mathcal{E}$ domain : $\mathcal{I}(G)^I$

Initial stack,: $\left[[S'\rightarrow\bullet S\perp, 1, 1]\middle|\perp\right]$

Final stack,: $\left[[S'\rightarrow S^{1,n+1}\bullet\perp, 1, n+1]\middle|\blacktriangle\right]$

Transitions :

PUSH : $[A\rightarrow\alpha\bullet B\beta, i, j], () \mapsto [B\rightarrow\bullet\eta, j, j], \varepsilon$

SWAP : $[A\rightarrow\alpha\bullet a\beta, i, j], a \mapsto [A\rightarrow\alpha a\bullet\beta, i, j+1], \varepsilon$

POP : $[A\rightarrow\alpha\bullet, i, j], [B\rightarrow\eta\bullet A\beta, k, i], () \mapsto [B\rightarrow\eta A^{i,j}\bullet\beta, k, j], A^{i,j}\rightarrow\alpha$

5.2 Earley PPDA Interpretation.

5.2.1 Synchronization Problem.

The absence of probabilities hides a problem that raises when computing them : as soon as a node is recognized, its parent can pursue its recognition, but computing the parent probabilities requires the probabilities of all its ambiguous children.

To continue not to take care of the computation order, symbolic probabilities can be introduced (representing probabilities of ambiguous children). The system is stable at the end of a complete computation, and can be solved then. But this would forbid to base a parsing decision on probabilities.

To permit this, we choose a intermediate solution : $\mathcal{E}$ is stratified and its computation is synchronized with the tokens. $\mathcal{E} = \bigcup_{j\in[0,n]} \mathcal{E}_j$, with $\mathcal{E}_j$ being the set of the Earley-items having j as second index. The level j+1 is computed after the level j is completed.

5.2.2 The PPDT Computation Process.

The synchronization splits the transition set in two parts : the intra-level transitions (PUSH and POP) and the inter-level ones (SWAP). The computation process is slightly modified.

Instead of having a single loop on the transitions set, there are two : one for inter-level (PUSH-POP), one for intra-level (SWAP). Each is executed until no new 1-items are produced and then the other starts a new cycle, until no transition is applicable or $\perp$ is reached.

PUSH and POP are applied as usual : the 1-items on the left (of the transition) are chosen in $\mathcal{E}$ and the one on the right is added to $\mathcal{E}$.

For SWAP, there is a difference : the 1-item on the right is added in set : $\mathcal{E}_{j+1}$, while $\mathcal{E} = \bigcup_{k=0}^{k=j} \mathcal{E}_k$; the 1-items on the left are chosen in $\mathcal{E}$.

When the inter-level loop does not add any new 1-items in $\mathcal{E}_{j+1}$, j is incremented.

5.2.3 Symbolic Probabilities.

$p(\bullet B_j)$ is introduced because the initial Earley-items with a B in LHS are produced by all the Earley-items having a dot in front of B in level j : there is such a symbol for each B and each j.

$$p(\bullet B_j) = \sum_{<A\to\alpha\bullet B\beta,i,j,\mathbf{P},\mathbf{R}>\in\mathcal{E}} \mathbf{P}$$

$p(A_{ij}\bullet)$ is introduced because a subtree rooted in $A^{i,j}$ and spanning $x_{i..j}$ may be produced more than once. Such a symbol may exist for each (A, i, j).

$$p(A_{ij}\bullet) = \sum_{<A\to\alpha\bullet,i,j,\mathbf{P},\mathbf{R}>\in\mathcal{E}} \mathbf{R}$$

By definition, these variables can be evaluated if the level j is complete, they form an invertible linear system. This is easy to prove with G consistency : let N be the vector of all non-terminals: $N_l \in \mathcal{N}$. The $p(\bullet B_j)$ solving problem amounts to finding the $p(\bullet N)$ vector s.t.

$$p(\bullet N)_l = \sum_{<N_r\to\alpha\bullet N_l\beta,i,j,\mathbf{P},\mathbf{R}>\in\mathcal{E}} \mathbf{P}$$

$$= \sum_{<N_r\to\alpha\bullet N_l\beta,i,j,\mathbf{P},\mathbf{R}>\in\mathcal{E},s.t.i\neq j} \mathbf{P} + \sum_{<N_r\to\alpha\bullet N_l\beta,j,j,\mathbf{P},\mathbf{R}>\in\mathcal{E}} \mathbf{P}$$

$= K_l + \sum_r L_{l,r}$ where K_l is the constant vector K l^{th} element and
$L_{l,r} = p_p([N_r\to\alpha\bullet N_l\beta, j, j]) = p_p([N_r\to\bullet\alpha N_l\beta, j, j])\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon) = p(\bullet N)_r\, p(N_r\to\alpha N_l\beta)\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon)$.

Thus $p(\bullet N) = M\, p(\bullet N) + K$, where $M_{l,r} = p(N_r \overset{1}{\rightsquigarrow}_l \alpha N_l\beta)\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon)$.

The equality $(M^p)_{l,r} = p(N_r \overset{p}{\rightsquigarrow}_l \alpha N_l\beta)\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon)$ is easy to verify. Hence, the consistency condition leads to: $\lim_{p\to\infty} (M^p)_{l,r} = 0$, therefore, there exists a κ s.t. for each $q \geq \kappa$, $\sum_l (M^q)_{l,r} < 1$, then $||M^q||_1 < 1$ [5]. Hence, the series $\sum_{i=0}^{\infty} M^i$ converges, *i.e.* $(I - M)$ is invertible, $p(\bullet N) = (I - M)^{-1} K$ and $(I - M)^{-1}_{l,r} = p(N_r \overset{}{\rightsquigarrow}_l \alpha N_l\beta)\, p(\alpha \overset{}{\rightsquigarrow}_l \epsilon) = p(N_r \overset{}{\rightsquigarrow}_l N_l\beta)$.

The $p(A_{ij}\bullet)$ system is similar, the corresponding matrix Q being s.t. $(I - Q)^{-1}_{l,r} = p(N_r \overset{}{\rightsquigarrow}_l N_l)$.

Important note: M and Q matrices can be statically inverted since they depend only on G.

5.2.4 Description of the PPDT.

$\mathcal{E}$ domain : $\mathcal{I}(G)^I \times \mathcal{D}(p) \times \mathcal{D}(p)$.

Initial stack : $[<S'\to\bullet S\perp, 1, 1, 1, 1> \mid \perp]$

Final stack : $[<S'\to S\bullet\perp, 1, n+1, \Pi, \Pi> \mid \perp]$

Intra-level :

PUSH

$<A\to\alpha\bullet B\delta, i, j, \mathbf{P}, \mathbf{R}>, () \mapsto <B\to\bullet\eta, j, j, p(\bullet B_j)\, P(B\to\eta), P(B\to\eta)>, \varepsilon$

POP

$<A\to\alpha\bullet, i, j, \mathbf{P}'', \mathbf{R}''>, <B\to\eta\bullet A\beta, k, i, \mathbf{P}, \mathbf{R}>, () \mapsto <B\to\eta A^{i,j}\bullet\beta, k, j, \mathbf{P}p(A_{ij}\bullet), \mathbf{R}p(A_{ij}\bullet)>, (A^{i,j}\to\alpha, \mathbf{R}'')$

Inter-level : **SWAP**

$<A\to\alpha\bullet a\beta, i, j, \mathbf{P}, \mathbf{R}>, a \mapsto <A\to\alpha a\bullet\beta, i, j+1, \mathbf{P}, \mathbf{R}>, \varepsilon$

[5] $||M||_1$ is then 1-norm defined by $\max_r \sum_l |M_{l,r}|$.

5.3 Proof indications.

PUSH prediction: a node N, that has never been visited, is reached. Contrarily to the deterministic computation, N can be reached from several positions: all the ones that have (at level j) a dot in front of B. That is why p_p is $p(lab(N))$ multiplied by $p(\bullet B_j)$.

POP: I,J $\mapsto$ K. K probabilities are still based on J ones, yet they do not only depend on $p_r(I)$ but on the sum of the probabilities of all the subtrees rooted in $A^{i,j}$: $p(A_{ij}\bullet)$.

6 Left Corner.

Earley's algorithm is completely dynamic: nothing is statically produced using the grammar before parsing. [Stolcke 93] gives a stochastic parse-tree recognition algorithm based on Earley's, but which avoids the dynamic resolution of our systems by solving them statically. After [Leermakers 89], we propose to integrate this optimization in a process which performs syntactic static computation: Left Corner (LC).

The LC strategy, seen as a tree exploration, is the same as Earley's. But all the prediction calls are statically computed: a finite automaton is compiled using the grammar.

6.1 Construction of the Underlying Finite Automaton.

Considering the augmented grammar $(\Sigma\cup\{\bot\}, \mathcal{N}\cup\{S'\}, \mathcal{R}\cup\{S'\rightarrow S\bot\}, S')$.

Given a non-initial dotted rule, we want all the initial dotted rules produced by an Earley prediction. For this we use the LC relation.

Definition 24. *Left Corner relation.*
$\angle = \{(A, B) \in \mathcal{N}^2 \mid B\rightarrow A\alpha \in \mathcal{R}\}$

Definition 25. *Left Corner state.*
s is a LC state iff $s = \{k(s)\} \cup nk(s)$ s.t. if $s = s_0$, the initial state, then $k(s_0) = S'\rightarrow\bullet S\bot$, else k(s) is a non initial dotted rule, and $nk(s) = \{C\rightarrow\bullet\alpha \mid k(s) = A\rightarrow X\beta\bullet B\gamma$ and $C \angle^* B\}$.
k(s) is the kernel of s, nk(s), its non-kernel set.

The set of LC states is $\mathcal{S}^{comp}$. It contains a single final state: $s_f = \{S'\rightarrow S\bullet\bot\}$.

Transitions are summarized in 3 tables. $s \in \mathcal{S}^{comp}$, $B \in \mathcal{N}$.

1. $GOTO_k(s, B) = r$ s.t. $k(s) = A\rightarrow\omega\bullet B\delta$ and $k(r) = A\rightarrow\omega B\bullet\delta$.
2. $GOTO_{nk}(s, B) = \{r \in \mathcal{S}^{comp} \mid A\rightarrow\bullet B\omega \in s$ and $k(r) = A\rightarrow B\bullet\omega\}$.
3. $ACTION(s, a) \in \mathcal{A}$, with $s \in \mathcal{S}^{comp}$, $a \in \Sigma$ and $\mathcal{A}$ is the set of actions.
   - ksc(r) (kernel scan) iff $k(s) = A\rightarrow\omega\bullet a\delta$ and $k(r) = A\rightarrow\omega a\bullet\delta$.
   - nksc(r) (non-kernel scan) iff $A\rightarrow\bullet a\omega \in s$ and $k(r) = A\rightarrow a\bullet\omega$
   - red($A\rightarrow\omega$) (reduction of $A\rightarrow\omega$) iff $A\rightarrow\omega\bullet \in s$.

By definition, a LC state contains all the possible paths from the kernel position to the leaves of all the trees where the kernel position can appear. During the parse, when moving from, say, s to r, one path is pursued among all the potential paths in s: until a transition is executed from a state, its real position is given by its kernel, the non-kernel set contains virtual positions.

The LC PDA does not need any prediction transition since it is already within the LC states, but scan and completion transitions have to manage the virtual paths becoming real: this explains the splitting kernel/non-kernel, corresponding to Earley-like/prediction management.

6.2 The LC PDA.

As a state is associated to the dotted rule in its kernel, (s, i, j) is associated to the corresponding Earley-item.

Stack alphabet $\mathcal{S}^{comp}\times[0, n]\times[0, n]\cup\{\bot\}$.

Initial stack : $[(s_0, 1, 1)|\bot]$

Final stack : $[(s_f, 1, n+1)|\bot]$

Scans :

- KERNEL SCAN.
  if ksc(r) $\in$ ACTION(s, a), apply
  SWAP (s, i, j), a $\mapsto$ (r, i, j+1).
- NON-KERNEL SCAN.
  if nksc(r) $\in$ ACTION(s, a), apply
  PUSH (s, i, j), a $\mapsto$ (r, j, j+1), (s, i, j).

Reductions :

- KERNEL REDUCE.
  - if red(A$\rightarrow$Xω) $\in$ ACTION(s, a) and r $\in$ GOTO$_k$(s', A), apply
    POP (s, i, j), (s', k, i), (a) $\mapsto$ (r, k, j).
  - if red(A$\rightarrow\epsilon$) $\in$ ACTION(s, a) and r $\in$ GOTO$_k$(s, A), apply
    SWAP (s, i, j), (a) $\mapsto$ (r, i, j).
- NON-KERNEL REDUCE.
  - red(A$\rightarrow$Xω) $\in$ ACTION(s, a) and r $\in$ GOTO$_{nk}$(s', A), apply
    SWAP2 (s, i, j), (s', k, i), (a) $\mapsto$ (r, i, j), (s', k, i).
  - if red(A$\rightarrow\epsilon$) $\in$ ACTION(s, a) and r $\in$ GOTO$_{nk}$(s, A), apply
    PUSH (s, i, j), (a) $\mapsto$ (r, j, j), (s, i, j).

6.3 Proof indications.

Replacing states by their kernels make the kernel transitions appear like in Earley PDA.

Non-kernel transitions correspond to taking in account the static predictions while shifting a symbol — terminal for scans, non-terminal for reductions. Each non-kernel transition nk-T deals with the same action than the kernel one k-T, but manages the realization of the prediction: nk-T shifts a symbol X in a non-kernel rule, the corresponding move in the tree is (1) getting down from the kernel, by performing a PUSH, and (2) shifting X, by SWAPing the pushed position by the shifted one. The nk-T directly pushes a shifted position.

A second splitting occurs: ϵ-reduction or not. Contrarily to a non-ϵ reduction the complete dotted rule is not a whole state that must be popped: the transition on the LHS is done without popping because its parent is in the same state.

6.4 LC PPDA.

LC parsing computes no prediction dynamically. It is so with probabilities. Each predicted position is statically computed with its (predicted) probability. As a LC state contains a part of shared forest, it will not be possible (or completely uninteresting) to extract the prediction probability of one tree. Hence, prediction probabilities will be computed only during the dynamic programming interpretation.

6.4.1 The Underlying Automaton.

Each time a rule appears with its first symbol recognized (*i.e.* in non-kernel transitions and ϵ-reductions) its probability must be communicated to the parser to compute the recognition probability. This justifies the following tables modifications:

- GOTO$_{nk}$(s, X) = {(r, p) | A$\rightarrow\bullet$Xω $\in$ s, A$\rightarrow$X$\bullet\omega$ $\in$ r and p = p(A$\rightarrow$Xω)}.
- Non-kernel scans: nksc(r, p) s.t. A$\rightarrow$X$\bullet\omega$ $\in$ r and p = p(A$\rightarrow$Xω)
- red(A$\rightarrow\epsilon$, p_ϵ): p_ϵ = p(A$\rightarrow\epsilon$).

6.4.2 The Automaton.

Stack alphabet : $\mathcal{S}^{comp} \times [0, n] \times [0, n] \times \mathcal{D}(\mathcal{P}) \cup \blacktriangle$

Initial stack $[(s_0, 1, 1, 1) | \bot]$

Final stack $[(s_f, 1, n+1, \Pi) | \bot]$

KERNEL SCAN : if ksc(r) $\in$ ACTION(s, a), apply
SWAP (s, i, j, $\mathbf{R}$), a $\mapsto$ (r, i, j+1, $\mathbf{R}$)

NON-KERNEL SCAN : if nksc(r, p) $\in$ ACTION(s, a), apply
PUSH (s, i, j, $\mathbf{R}$), a $\mapsto$ (r, j, j+1, p), (s, i, j, $\mathbf{R}$)

KERNEL REDUCE :

- if red(A$\rightarrow$Xω) $\in$ ACTION(s'', a) and r $\in$ GOTO$_k$(s, A), apply
  POP (s'', i, j, $\mathbf{R}''$), (s, k, i, $\mathbf{R}$), (a) $\mapsto$ (r, k, j, $\mathbf{R}\mathbf{R}''$)
- if red(A$\rightarrow\epsilon$, p_ϵ) $\in$ ACTION(s, a) and r $\in$ GOTO$_k$(s, A), apply
  SWAP (s, i, j, $\mathbf{R}$), (a) $\mapsto$ (r, i, j, $\mathbf{R}p_\epsilon$)

NON-KERNEL REDUCE :

- if red(A$\rightarrow$Xω) $\in$ ACTION(s'', a) and (r, p) $\in$ GOTO$_{nk}$(s, A), apply
  SWAP2 (s'', i, j, $\mathbf{R}''$), (s, k, i, $\mathbf{R}$), (a) $\mapsto$ (r, i, j, p$\mathbf{R}''$), (s, k, i, $\mathbf{R}$)
- if red(A$\rightarrow\epsilon$, p_ϵ) $\in$ ACTION(s, a) and (r, p) $\in$ GOTO$_{nk}$(s, A), apply
  PUSH (s, i, j, $\mathbf{R}$), (a) $\mapsto$ (r, j, j, pp_ϵ), (s, i, j, $\mathbf{R}$)

6.5 Proof indications.

As for the PDA, only non-kernel transitions and ϵ-reductions are to be verified — the others being the same as Earley PPDA.

The non-kernel scan corresponds to recognizing a terminal from an initial position I : the prediction probability is $\underset{p}{P}(I) = p$, given by nksc(r,p).

For reductions, the probability of the recognized subtree t must be multiplied to the previous position, as usual. In case of ϵ-reduction, $\underset{r}{P}(t) = p_\epsilon$, given by ACTION, otherwise by the first popped 1-item ($\mathbf{R}''$). In case of non-kernel reduction, the recognition probability of the previous position is given by GOTO$_{nk}$ (p), otherwise by the second popped 1-item ($\mathbf{R}$).

7 LC Dynamic Programming Interpretation.

7.1 LC PDT.

To produce the shared forest, we need to introduce an extra parameter in the 1-items : the decorated shifted part of the kernel RHS.

$\mathcal{E}$ domain : $\mathcal{S}^{comp} \times [0, n] \times [0, n] \times \mathcal{W}^*$

Initial stack : $[(s_0, 1, 1, \epsilon) | \bot]$

Final stack : $[(s_f, 1, n+1, S^{1,n+1}) | \bot]$

Transitions :

Scans :

- KERNEL SCAN. if ksc(r) $\in$ ACTION(s, a), apply
  SWAP : (s, i, j, α), a $\mapsto$ (r, i, j+1, αa), ε.
- NON-KERNEL SCAN. if nksc(r) $\in$ ACTION(s, a), apply
  PUSH : (s, i, j, α), a $\mapsto$ (r, j, j+1, a), ε.

Reductions

- KERNEL REDUCE.
  - if red(A→Xω) ∈ ACTION(s, a) and r ∈ $GOTO_k$(s′, A), apply POP : (s, i, j, α), (s′, k, i, β), (a) ↦ (r, k, j, $\beta A^{i,j}$), $A^{i,j}$→α.
  - if red(A→ϵ) ∈ ACTION(s, a) and r ∈ $GOTO_k$(s, A), apply SWAP : (s, i, j, α), (a) ↦ (r, i, j, $\alpha A^{j,j}$), $A^{j,j}$→ϵ.
- NON-KERNEL REDUCE.
  - if red(A→Xω) ∈ ACTION(s, a) and r ∈ $GOTO_{nk}$(s′, A), apply SWAP2 : (s, i, j, α), (s′, k, i, β), (a) ↦ (r, i, j, $A^{i,j}$), $A^{i,j}$→α
  - if red(A→ϵ) ∈ ACTION(s, a) and r ∈ $GOTO_{nk}$(s, A), apply PUSH : (s, i, j, α), (a) ↦ (r, j, j, $A^{j,j}$), $A^{j,j}$→ϵ.

7.2 PPDA Interpretation.

7.2.1 The Underlying Automaton.

Definition 26. *Dotted rule prediction probability.*
Let s be a LC state, p_p(A→α•β | s) = p_p([A→α•β, i, j]) / p_p(k(s), i, j)).

Hence, p_p(k(s) | s) = 1 and p_p(A→•Bα | s) = p(A→Bα) p_s(•B),
where p_s(•B) = $\sum_{C\to\omega\bullet B\delta\in s} p_p(C\to\omega\bullet B\delta|s)$. [6]

As for p(•B_j), the p_s(•B) form a invertible linear system, that is then statically inverted.

Only non-kernel transitions are changed : they have an extra parameter to take in account the statically computed prediction probabilities.

Let (r, p, π) be in a non-kernel transition from s on X. Then A→•Xω ∈ s, k(r) = A→X•ω and π = p_p(A→•Xω | s).

7.2.2 The Symbolic Probabilities.

For Earley strategy, we introduced two kinds of variable symbol : p(•B_j) for prediction, p(A_{ij}•) for completion. The last one is simply adapted to LC states, the prediction adaptation is less immediate because predictions are made during non-kernel transitions.

- p(•r_j) is the prediction probability of r kernel rule in its initial position at level j.

$$p(\bullet r_j) = \sum_{\substack{(s,i,j,\mathbf{P},\mathbf{R},\alpha)\in\mathcal{E}\\ s.t.\\ (r,p,\pi)\in GOTO_{nk}(s,A)\\ or\\ nksc(r,p,\pi)\in ACTION(s,x_j)}} \mathbf{P}\pi$$

- The ϵ-reductions force us to make a particular case.

  - i ≠ j :

$$p(A_{ij}\bullet) = \sum_{\substack{(s'',i,j,\mathbf{P}'',\mathbf{R}'',\alpha)\in\mathcal{E}\\ s.t.\ red(A\to\omega)\in ACTION(s'',x_j)}} \mathbf{R}''$$

  -

$$p(A_{jj}\bullet) = \sum_{\substack{(s,i,j,\mathbf{P},\mathbf{R},\alpha)\in\mathcal{E}\\ s.t.\ red(A\to\epsilon,p_\epsilon)\in ACTION(s,x_j)}} p_\epsilon$$

Without ϵ-rules, values are just incrementally accumulated, without needing any matrix inversion. Anyway we saw in the invertibility proof, section 5.2.3, that the inversions can be performed statically.

[6] p_s(•B) appears in [Wright&Wrigley 89] for a probabilistic LR parser, it is noted P_B.

7.2.3 LC PPDT.

$\mathcal{E}$ domain: $\mathcal{S}^{comp} \times [0, n] \times [0, n] \times \mathcal{D}(\mathcal{P}) \times \mathcal{D}(\mathcal{P}) \times \mathcal{W}^*$

Initial stack: $[(s_0, 1, 1, 1, 1, \epsilon) | \bot]$

Final stack: $[(s_f, 1, n+1, \Pi, \Pi, S^{1,n+1}) | \blacktriangle]$

Inter-level, scans:

- KERNEL SCAN: if ksc(r) $\in$ ACTION(s, a), apply
  SWAP (s, i, j, $\mathbf{P}$, $\mathbf{R}$, α), a $\mapsto$ (r, i, j+1, $\mathbf{P}$, $\mathbf{R}$, αa), ε
- NON-KERNEL SCAN: if nksc(r, p, π) $\in$ ACTION(s, a), apply
  PUSH (s, i, j, $\mathbf{P}$, $\mathbf{R}$, α), a $\mapsto$ (r, j, j+1, $\mathcal{P}(\bullet r_j)$, p, a), ε

Intra-level, reductions:

- KERNEL REDUCE:
  - if red(A$\rightarrow X\omega$) $\in$ ACTION(s'', a) and r $\in$ GOTO$_k$(s, A), apply
    POP (s'', i, j, $\mathbf{P}''$, $\mathbf{R}''$, α), (s, k, i, $\mathbf{P}$, $\mathbf{R}$, β), (a) $\mapsto$ (r, k, j, $\mathbf{P}\mathcal{P}(A_{ij}\bullet)$, $\mathbf{R}\mathcal{P}(A_{ij}\bullet)$, $\beta A^{i,j}$), ($A^{i,j}\rightarrow\alpha$, $\mathbf{R}''$).
  - if red(A$\rightarrow\epsilon$, p_ϵ) $\in$ ACTION(s, a) and r $\in$ GOTO$_k$(s, A), apply
    SWAP (s, i, j, $\mathbf{P}$, $\mathbf{R}$, α), (a) $\mapsto$ (r, i, j, $\mathbf{P}\mathcal{P}(A_{jj}\bullet)$,$\mathbf{R}\mathcal{P}(A_{jj}\bullet)$, $\alpha A^{j,j}$), ($A^{j,j}\rightarrow\epsilon$, p_ϵ).
- NON-KERNEL REDUCE:
  - if red(A$\rightarrow$Xω) $\in$ ACTION(s'', a) and (r, p, π) $\in$ GOTO$_{nk}$(s', A), apply
    SWAP2 (s'', i, j, $\mathbf{P}''$, $\mathbf{R}''$, α), (s, k, i, $\mathbf{P}$, $\mathbf{R}$, β), (a) $\mapsto$ (r, i, j, $\mathcal{P}(\bullet r_i)\mathcal{P}(A_{ij}\bullet)$, $p\mathcal{P}(A_{ij}\bullet)$, $A^{i,j}$), ($A^{i,j}\rightarrow\alpha$, $\mathbf{R}''$).
  - if red(A$\rightarrow\epsilon$, p_ϵ) $\in$ ACTION(s, a) and (r, p, π) $\in$ GOTO$_{nk}$(s, A), apply
    PUSH (s, i, j, $\mathbf{P}$, $\mathbf{R}$, α), (a) $\mapsto$ (r, j, j, $\mathcal{P}(\bullet r_j)\mathcal{P}(A_{jj}\bullet)$, $p\mathcal{P}(A_{jj}\bullet)$, $A^{j,j}$), ($A^{j,j}\rightarrow\epsilon$, p_ϵ).

7.3 Proof indications.

Non-kernel scan: the prediction probability is exactly $\mathcal{P}(\bullet r_j)$ definition. Because of r kernel, the subtree identified by (r, j, j+1) has a single node, and p is its probability.

Kernel-ϵ-reduction: w.r.t. the PPDA, p_ϵ is replaced by $\mathcal{P}(A_{jj}\bullet)$, which sums the probability of all the subtrees rooted in $A^{j,j}$.

Non-kernel non-ϵ-reductions: in a similar way $\mathcal{P}(A_{ij}\bullet)$ replaces the prediction probability of the reduced 1-item. W.r.t. Earley, $\mathcal{P}(\bullet r_j)$ summarizes the prediction probabilities of the Earley-items of the form [A$\rightarrow\bullet$Bβ, i, j].

8 Conclusion.

Like [Stolcke 93], we have presented an Earley-like stochastic parser that computes prefix and sub-parse-trees probabilities. The stochastic shared forest is produced with the subtree probability on each node.

Under Stolcke's first assumption (no ϵ-rules), our LC algorithm improves his, since the LC states contain all the necessary prediction probabilities. The improvement is less important without this assumption, but [Schabes 91] ideas on ϵ-rules optimization give back its interest to LC strategy (w.r.t. Earley): it consists in computing the ϵ-reductions statically, within the LC states (probabilities just follow) hence no prediction loops remain to be managed during parsing.

The most likely (Viterbi) parse can be easily obtained by replacing the + operator, in the probability domain, by *max* (loops are not to be followed since they lower the probability).

More formally, we have showed that Earley and LC strategies amount to a left-to-right depth-first exploration of the parse trees. It justifies the prediction probability definition as a left derivation probability, what permits to see it as a prefix probability.

More generally, we adopted the formal languages and automata theory point of view. Algorithms are presented within a framework that naturally generalizes the purely syntactic ones and their computations are proved correct w.r.t. the definitions.

References

[Booth-Thompson 73] Taylor L. Booth, Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transaction on Computers*, C-22(5) pp 442–450, 1973.

[Earley 70] Jay Earley. An efficient context-free parsing algorithm. Communications of the ACM 6. pp 451–455, 1970.

[Lang 74] Bernard Lang. Deterministic techniques for efficient non-deterministic parsers. Editor J. Loeckx. *Procedings of the Second Colloquium on Automata, Languages and Programming*, vol. 14 of *Lectures Notes in Computer Science*, pp 255–269. Springer-Verlag, 1974.

[Lang 89] Bernard Lang. Towards a uniform formal framework for parsing. Current issues in parsing technology, M.Tomita ed., Kluver Academic Publisher. 1991.

[Leermakers 89] R. Leermakers. How to cover a grammar. 27^{th} *meeting of the association for computational linguistics*, Vancouver, 1989.

[Schabes 91] Yves Schabes. Polynomial time and space shift-reduce parsing of arbitrary context-free grammars. 29^{th} meeting of the association for computational linguistics, Berkeley, 1991.

[Stolcke 93] Andreas Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. TR-93-065, Berkeley, 1993.

[Tendeau 95] Frédéric Tendeau. Stochastic parse-tree recognition by a pushdown automaton. To appear as an INRIA-Rocquencourt research report, 1995.

[VdlC 93] Éric Villemonte de la Clergerie. Automates à pile et programmation dynamique - DyALog : une application à la programmation en logique. *PhD thesis, Université Paris VII*, 1993.

[Wright&Wrigley 89] J.H Wright, E.N. Wrigley. Probabilistic LR parsing for speech recognition. *Proceedings IWPT*, pp 105–114, 1989.