A probabilistic top-down parser for minimalist grammars

T. Mainguy

this paper was written during an internship
supervised by E. Stabler (UCLA) and O. Catoni (ENS)

September 21, 2010

Abstract

This paper describes a probabilistic top-down parser for minimalist grammars. Top-down parsers have the great advantage of having a certain predictive power during the parsing, which takes place in a left-to-right reading of the sentence. Such parsers have already been well-implemented and studied in the case of Context-Free Grammars (see for example [Roa97]), which are already top-down, but these are difficult to adapt to Minimalist Grammars, which generate sentences bottom-up. I propose here a way of rewriting Minimalist Grammars as Linear Context-Free Rewriting Systems, allowing us to easily create a top-down parser. This rewriting allows also to put a probabilistic field on these grammars, which can be used to accelerate the parser. I propose also a method of refining the probabilistic field by using algorithms used in data compression.

Contents

Throughout this paper, I will refer as a *subtree* of a tree $\mathcal{T}$, the set of nodes in $\mathcal{T}$ dominated by a particular node, which will be the root of the subtree. On the other hand, a cut is the set of the leaves of a finite prefix tree of $\mathcal{T}$.

1 Introduction

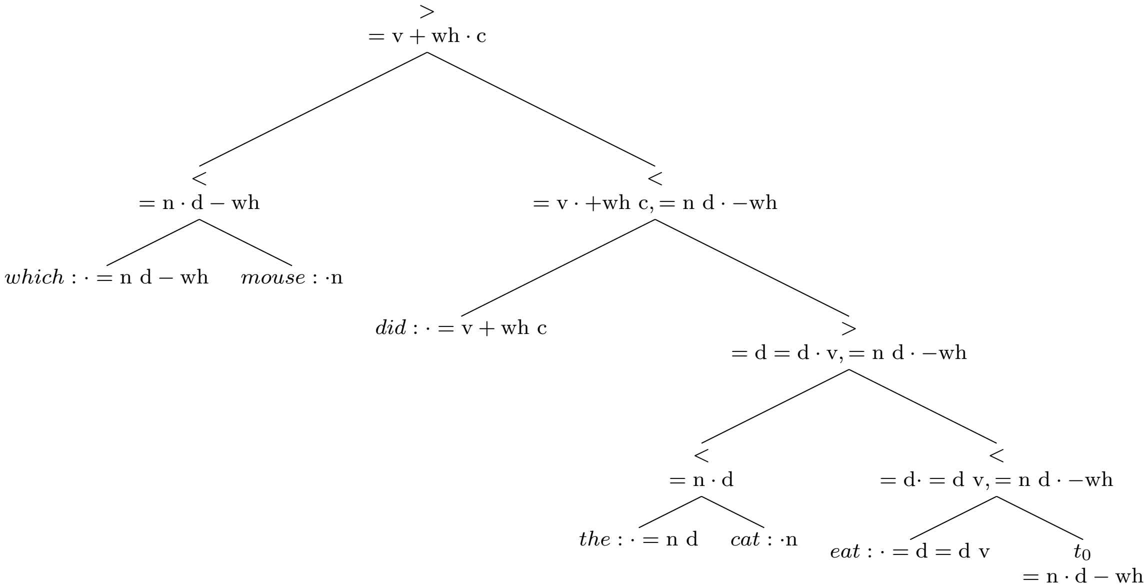
The idea of this parser is to see a minimalist grammar (MG) as a linear context-free rewriting system (LCFRS) on its derivation trees. This transformation allows us to work on a grammar without movement, generating sentences from top to bottom (on contrary of MG, which generates sentences bottom-up), and to put a probabilistic field on it.

1.1 Minimalist grammars

Minimalist grammars are designed to generate (subparts of) human natural languages. They are framed in Chomsky's minimalist program [Cho95], and were first described by E. Stabler in [Sta97]. For the sake of clarity, I will in this paper use slightly different convention to represent the trees generated by a minimalist grammar.

Minimalist grammars distinguishe themselves from more classical context-free grammars by the fact that they allow *movement*, commonly required by syntacticians to generate such sentences as (for example) 'Which mouse did the cat eat', where 'which mouse' is base-generated at the end of the sentence (in the object position), and moves at the front. The tree corresponding to this sentence, as generated by the toy Minimalist Grammar we will consider here as example, is the following:

(1)


$>$
$= \mathrm{v} + \mathrm{wh} \cdot \mathrm{c}$
$<$
$= \mathrm{n} \cdot \mathrm{d} - \mathrm{wh}$
$<$
$= \mathrm{v} \cdot +\mathrm{wh}\ \mathrm{c}, = \mathrm{n}\ \mathrm{d} \cdot -\mathrm{wh}$
$\textit{which} : \cdot = \mathrm{n}\ \mathrm{d} - \mathrm{wh}$
$\textit{mouse} : \cdot \mathrm{n}$
$\textit{did} : \cdot = \mathrm{v} + \mathrm{wh}\ \mathrm{c}$
$>$
$= \mathrm{d} = \mathrm{d} \cdot \mathrm{v}, = \mathrm{n}\ \mathrm{d} \cdot -\mathrm{wh}$
$<$
$= \mathrm{n} \cdot \mathrm{d}$
$<$
$= \mathrm{d} \cdot = \mathrm{d}\ \mathrm{v}, = \mathrm{n}\ \mathrm{d} \cdot -\mathrm{wh}$
$\textit{the} : \cdot = \mathrm{n}\ \mathrm{d}$
$\textit{cat} : \cdot \mathrm{n}$
$\textit{eat} : \cdot = \mathrm{d} = \mathrm{d}\ \mathrm{v}$
t_0
$= \mathrm{n} \cdot \mathrm{d} - \mathrm{wh}$

where t_0 denotes the *trace* of the subtree 'which mouse', which has moved in front of the sentence. A trace is kept for psychological reasons, as these traces can be shown to be still present for the computation of the meaning of sentences. They also allows to keep what is called the *deep structure*, corresponding to the tree where no movement happened, and all constituents are in their base position, where lexical selection takes place.

A minimalist grammar takes several *lexical elements*, and builds a tree with them. The toy grammar we consider will have the following lexical items:

(2) •*mouse* :: n
•*cat* :: n
•*the* ::= n d
•*which* ::= n d − wh
•*ate* ::= d = d c
•*eat* ::= d = d v
•*did* ::= v + wh c
•*did* ::= v c

This grammar generates (roughly) all affirmative/interrogative past sentences about a cat and a mouse eating each other.

As can be seen, many symbols are used next to the actual *phonetic contents* of the words (the, eat, cat, etc...). These are *syntactic features*, and the sequence of these in a lexical item represents its *syntactic category*, and is all that is needed to compute the tree. Two lexical items with the same lexical category can be freely interchanged without losing grammaticality.

The *syntactic features* may be of four types:

- *categories*, represented by a string of letters, among which is the *distinguished feature* c, used to recognise the grammatical outputs. For example, n. The set of categories will be noted Cat.
- *selectors*, represented by the string of letters of a category, preceded by a =. For example, =n. The set of selectors will be noted Sel.
- *licensees*, represented by a string of letters preceded by a -. For example, -wh. The set of licensees will be noted Licensee.
- *licensors*, represented by a string of letters corresponding to a licencee, preceded by a +. For example, +wh. The set of licensors will be noted Licensor.

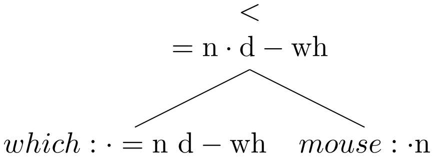
Syntactic features must follow a certain order, to ensure good formation of trees : Syn = $(\mathrm{Select}(\mathrm{Select} \cup \mathrm{Licensor})^{*})\mathrm{Cat}\mathrm{Licensee}^{*}$

The trees are computed by using two functions on the lexical items, to form *constituents* (i.e., trees):

- *merge*, when a selector selects a corresponding category,
- *move*, when a licensee moves to a corresponding licensor.

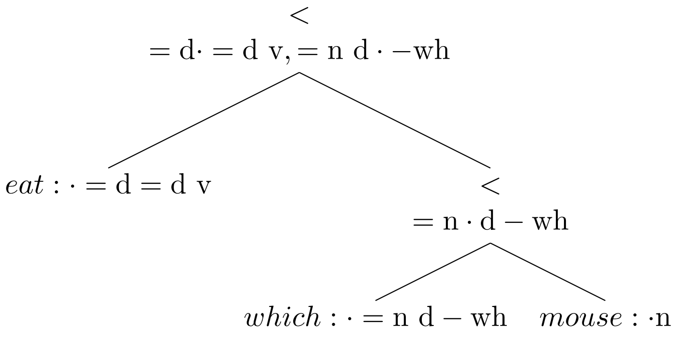
Let's see how this works on our little tree:

- Take *which* : · = n d − wh and *mouse* : ·n. We added a · in front of the syntactic categories to keep track of the derivation. Here, the two features just right of the dot (the *current features* are =n and n. It's a selector and its corresponding category, so we can *merge* them to a bigger constituent:

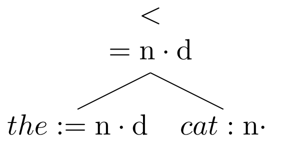


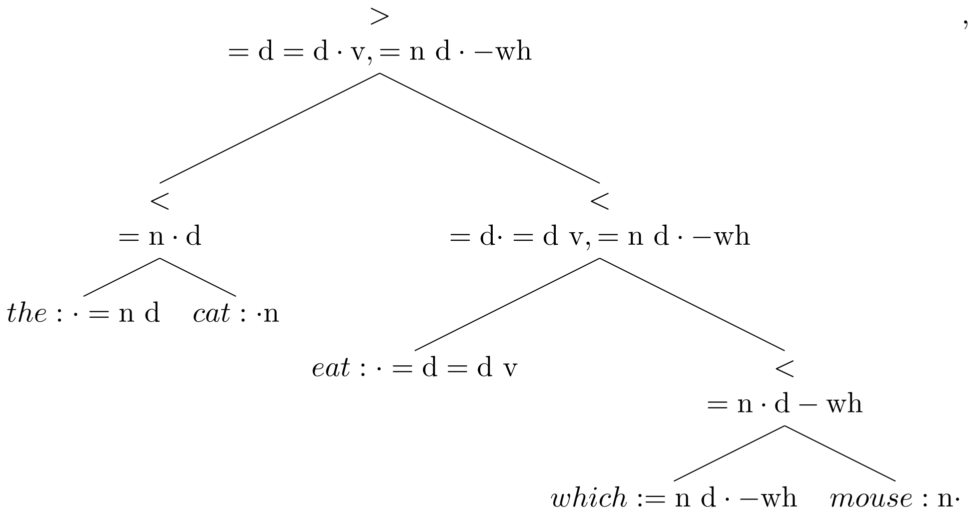
Both of the syntactic categories are copied to the root of the new constituent, with their dots moved one step right, since the current features were used. The category with the selector always comes first. If, as it is the case for the syntactic category of mouse, the dot ends up at the far right, the category may be left out, since it won't have any role in the further derivations. The $<$ indicates the *head* of the constituent, i.e. the constituent where the selector came from.

- Then merging the new constituent, whose syntactic category is $= \mathrm{n} \cdot \mathrm{d} - \mathrm{wh}$, with *eat* : $\cdot = \mathrm{d} = \mathrm{dv}$ (note the selector =d, corresponding to the current d) gives:



Note also that here, the second syntactic category still has something right of the dot, so stays.

- Together with , it merges into:

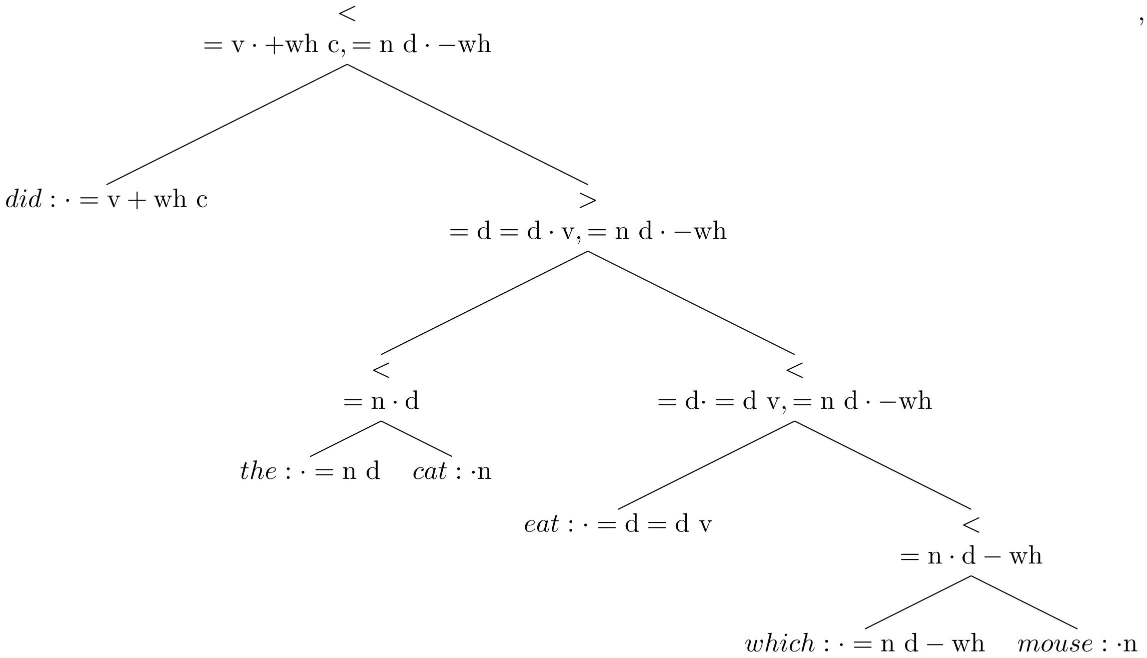


,

Note that for merging, only the first category in the list of syntactic categories is considered. Here, in $= \mathrm{d}\cdot = \mathrm{d}\ \mathrm{v}, = \mathrm{n}\ \mathrm{d} \cdot -\mathrm{wh}$, only $= \mathrm{d}\cdot = \mathrm{d}\ \mathrm{v}$ is considered (in fact, only $\cdot = \mathrm{d}$).

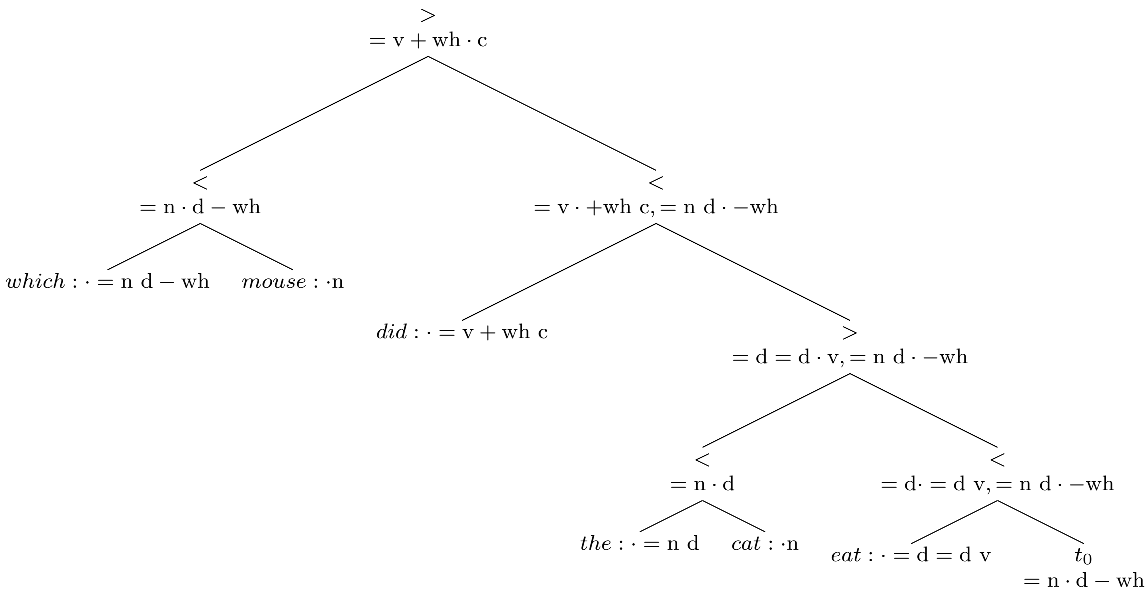
Note also that if the constituent with the selector is *complex* (i.e. is not formed of a single lexical item), as it is here the case, the merging happens in the other way: head right and selected constituent left. This has to do with the fact that english is a SVO language.

- It can then merge with *did* : · = v + wh c, giving:



,

- Finally, we can apply *move*. this function applies to a *single constituent*, whose first syntactic category has a *licensor* right of the dot. Here, +wh. It will then scan the other categories to find a corresponding licensee right of a dot (here, ·-wh), and move the corresponding constituent to the top of the tree, giving the final sentence:



The dots are moved as usual, the > indicates the constituent where the licensor was, and in this case, since the only feature right of a dot is the distinguished feature c, we know that the derivation yielded a grammatical output.

> The constituent moved corresponds to the biggest constituent whose head is the lexical element containing the considered lincensee (this corresponds to the syntactic notion of maximal projection).

The phonetical content of this sentence is the concatenation of the phonetic contents of its leaves, in left-to-right reading: ‘which mouse did the cat eat’.

The notion of *head* is an important notion in linguistics, since, by the principle of *locality of selection*, we want to restrict the amount of information that an item has access to. As such, the only information about a constituent accessible from outside (for a merge operation, for example), is the right-of-the-dot features of its head, that is, the features of the first syntactic category (minus the left-of-the-dot ones, kept only for the sake of historic bookkeeping).

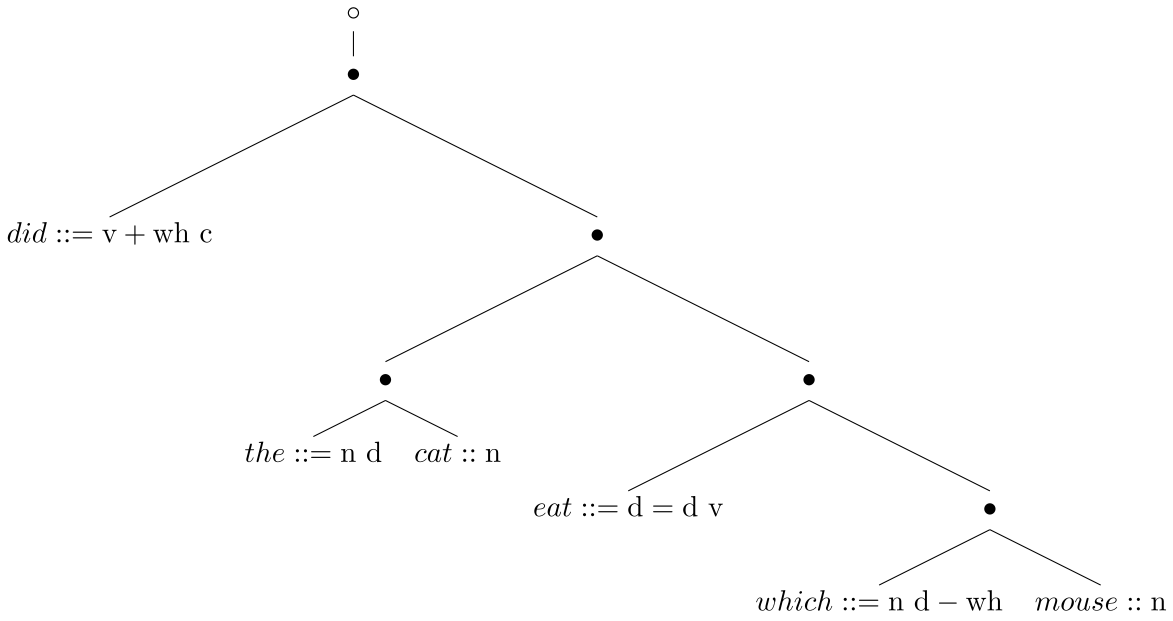
1.2 Derivation trees

Another way of representing the constituents generated by a grammar is by using its *derivation tree*:

Definition 1.1. *The* derivation tree *of a constituent is a binary tree showing the history of its building by the functions merge and move. Its leaves are lexical items, and its nodes are labelled by either* • *(merge, it's then a binary node) or* ○ *(move, it's then a unary node).*

For example, the derivation tree of the previous example (1), is:

(3)



Let's note that to each subtree of a derivation tree corresponds a unique constituent, appearing in the construction of the main one. We can then label each node of a derivation tree by the syntactic category of the corresponding constituent.

2 Probabilistic minimalist grammars

2.1 Derivation trees of MG as LCFRS-derived trees

The basis of this method is to see minimalist derivation trees as trees generated by linear context-free rewriting systems (LCFRS). Putting a probability field on these is indeed very easy. A similar approach was also used in the minimalist parser of H. Harkema in his thesis [Har01].

We thus take a general *minimalist grammar* $\mathcal{G} = (\sigma, \mathrm{Feat}, \mathrm{Lex}, \mathcal{F})$.

The closure of Lex under $\mathcal{F}$ gives the outputs of the grammars. Here, we will not consider these outputs, but the derivation trees describing the process giving these outputs.

One important difference between context-free grammars, for which probabilistic versions are well-studied, and minimalist grammars is that CFG generate trees from top to bottom, by means of rules rewriting each non-terminal node by a number of other nodes, while a MG generates trees from bottom to the top, by merging and moving elements. While in the CFG case, we begin with a single symbol and then choose rules to rewrite it (thus enabling us to assign probabilities to the process by assigning probabilities to the rewriting rules), in MG, we begin with a bunch of lexical items, not necessarily compatible with each other, and merge them together (and occasionally moving them too). Here we will present a way of seeing the generating process of MG as a LCFRS, which, as CFG, generates from top to bottom with a set of rules. The differents non-terminal symbols will be defined by closure of a certain set of axioms (starting symbols) under a set of inference rules, giving this way a top-down way of generating derivation trees of MG.

2.1.1 Categories and partial outputs

In order to do this, we will first define a particular type of objects, called categories, which will be the non-terminal symbols of our LCFRS:

Definition 2.1. *A category is either a lexical item, or a sequence of the form* $[\gamma_0 \cdot \delta_0, \ldots, \gamma_k \cdot \delta_k]$*, where* $\gamma_0, \ldots, \gamma_k, \delta_0, \ldots, \delta_k$ *are elements of Syn, or a special symbol start. A* simple *category is a category with its first dot at the leftmost place (and* $k = 0$*). Otherwise, it is a* complex *category. start is neither simple or complex.*

Categories corresponds exactly to the list of syntactic categories defined in 1.1, although our definition allows here categories which cannot be generated by a Minimalist Grammar. We will of course only be interested in those who are.

We then define a *partial output* as a string $\Delta_1 \ldots \Delta_n$ of categories. These represent a particular stage in the construction of a minimalist derivation tree by the corresponding LCFRS, the different categories being the categories of the partial derivation tree which is build.

2.1.2 Axiom

There is a single axiom, the category *start*.

2.1.3 Inference rules

These rules correspond to the rewriting rules of the Linear Context-Free Rewriting System $\mathcal{S}$ corresponding to our Minimalist Grammar $\mathcal{G}$. For each possible application of one of the functions *merge* or *move* of grammar $\mathcal{G}$ giving a particular category Δ, there is a corresponding inference rule (which gives quite a lot of rules...). Then, given a particular category Δ, the rules will tell how this particular type of tree (remember that categories describe a particular type of trees generated by the grammar) can be un-merged or un-moved into one (in case of un-move) or two (in case of un-merge) different types of trees. To this must be added the rules expanding the *start* category, and the lexicalisation rules. The first allows us to begin with a unique symbol, instead of all categories ending with the distinguished feature. The second ones allow us to leave the lexical part of the parsing up to the last moment. Thus here we are:

1. Start rules: for every lexical item $\gamma :: \delta\ \mathrm{c}$,
   Start:
   $\mathit{start} \longrightarrow [\delta \cdot \mathrm{c}]$
2. Re-writing rules for complex categories:
   (a) Un-merge rules: the left-hand category is of the form $[\delta = \mathrm{x} \cdot \beta, S]$
      i. Cases where the selector was a simple tree ($\delta = \epsilon$):
         A. For any lexical item of feature string $\gamma\ \mathrm{x}$,
            Unmerge-1:
            $[= \mathrm{x} \cdot \beta, S] \longrightarrow [\cdot = \mathrm{x}\ \beta][\gamma \cdot \mathrm{x}, S]$
         B. For any element $(\gamma\ \mathrm{x} \cdot \varphi) \in S$, with $S' = S - (\gamma\ \mathrm{x} \cdot \varphi)$,
            Unmerge-3, simple:
            $[= \mathrm{x} \cdot \beta, \gamma\ \mathrm{x} \cdot \varphi, S'] \longrightarrow [\cdot = \mathrm{x}\ \beta][\gamma \cdot \mathrm{x}\ \varphi, S']$
            It should be noted that necessarily, $\varphi \neq \emptyset$.
      ii. Cases where the selector was a complex tree:
         A. For any decomposition $S = U \sqcup V$, and any lexical item of feature string $\gamma\ \mathrm{x}$,
            Unmerge-2:
            $[\delta = \mathrm{x} \cdot \beta, S] \longrightarrow [\delta \cdot = \mathrm{x}\ \beta, U][\gamma \cdot \mathrm{x}, V]$
         B. For any element $(\gamma\ \mathrm{x} \cdot \varphi) \in S$, and any decomposition $S = U \sqcup V \sqcup (\gamma\ \mathrm{x} \cdot \varphi)$,
            Unmerge-3, complex:
            $[\delta = \mathrm{x} \cdot \beta, \gamma\ \mathrm{x} \cdot \varphi, S'] \longrightarrow [\delta \cdot = \mathrm{x}\ \beta, U][\gamma \cdot \mathrm{x}\ \varphi, V]$
            As in 2(a)iB, φ has to be non empty.
   (b) Un-move rules: the left-hand category is of the form $[\delta + \mathrm{f} \cdot \beta, S]$
      i. For any $(\gamma - \mathrm{f} \cdot \varphi) \in S$ (necessarily unique by the Shortest Movement Constraint), with $S' = S - (\gamma - \mathrm{f} \cdot \varphi)$,
         Unmove-2:
         $[\delta' + \mathrm{f} \cdot \beta, \gamma - \mathrm{f} \cdot \varphi, S'] \longrightarrow [\delta' \cdot +\mathrm{f}\ \beta, \gamma \cdot -\mathrm{f}\ \varphi, S']$
      ii. If there is no $(\gamma - \mathrm{f} \cdot \varphi) \in S$, then for any lexical item of feature string $\gamma - \mathrm{f}$,
         Unmove-1:
         $[\delta' + \mathrm{f} \cdot \beta, S] \longrightarrow [\delta' \cdot +\mathrm{f}\ \beta, \gamma \cdot -\mathrm{f}, S]$
3. Re-writing rules for simple categories: for any lexical item $\lambda :: \beta$,
   Lexicalize:
   $[\cdot \beta] \longrightarrow \lambda :: \beta.$

The set of *relevant partial outputs* can thus be defined as the closure of the axiom *start* under the inference rules. This set describes exactly all possible partial outputs given by the LCFRS $\mathcal{S}$, i.e. all possible strings of categories obtained by a cut through a tree generated by the LCFRS $\mathcal{S}$. Such a string correspond to a selection of outputs (not necessarily complete) generated by the minimalist grammar $\mathcal{G}$, such that they can be put together by application of *merge* and *move*, in the same order (two categories will get merged only if they are adjacent in the string) to obtain a complete output. A relevant output is a relevant partial output consisting of only lexical items. It corresponds to grammatical sentences.

The *relevant categories* are exactly the categories that appear in a relevant partial output. They correspond to the possible sets of similar partial trees generated by the grammar $\mathcal{G}$. They are in finite number, since, by the Shortest Movement Constraint, no two identical licensees can appear in the feature strings of a relevant category (omitting the first string). Thus two identical feature strings (diverging only by the position of the dot) can't appear together, and therefore the total length of all the feature strings of a relevant category is bounded by the sum of the length of all the feature strings of the lexical items, which is finite.

2.1.4 Derivation trees

With this formalism, we have now a quite straightformard way of defining minimalist derivation trees, in a way that enables us to put very simply probabilities on them: they are just the trees obtained by maximal application of rewriting rules to the axiom *start*. The probability is simply given by a probability field on the rules.

2.2 Probabilities on MG derivation trees

To define a probability field on the derivation trees of a MG, we now just have to put conditional probabilities on the rules discussed before, given the initial relevant category. The probability of a given tree will then be the product of the probabilities of the rules that generate it, as for regular probabilistic linear context-free rewriting systems. There can be however quite a lot of such rules and relevant categories, even if the MG is quite simple, but they can all be computed beforehand with the only knowledge of the grammar, thanks to the definition by closure of these categories. Indeed, we will see a simple method permitting to compute both the relevant categories and the inference rules that are needed.

It should be noted that the functions (Merge-1,2,3 and Move-1,2) having potentially given birth to a given relevant category are quite few (at most two), *only if we use the dot notation*, which keeps track of a minimal part of the history of the derivation. This is why the relevant categories should include all features of the lexical item potentially heading the tree (and not just the ones on the right of the dot).

To settle things a bit, we will here illustrate this method with a little example.

2.3 Example : $a^n b^n$

We will here consider the MG with the following lexical items (ϵ being the empty string):

(4) $\bullet \epsilon :: \mathrm{c}$

$\bullet \epsilon :: = \mathrm{a} + \mathrm{m}\ \mathrm{c}$

$\bullet a :: = \mathrm{b}\ \mathrm{a} - \mathrm{m}$

$\bullet b :: \mathrm{b}$

$\bullet b :: = \mathrm{a} + \mathrm{m}\ \mathrm{b}$

This grammar generates exactly the strings of the form $a^n b^n$, $n \in \mathbb{N}$. Since this is a context-free language, we wouldn't have needed to use licensors and licencees, but for the sake of getting a language simple enough with enough rules (especially movement ones), we will work on this one.

We now want to get the relevant categories of this language, and the corresponding 'context-free rules'. A quite straightforward way to obtain them is to start from the axiom *start* and follow the inference rules to close the set of relevant categories. From *start* we apply the schemes to get all applicable rules, apply them, get some new relevant categories, apply the schemes to get new rules, apply them, etc... Since they are in finite number, this algorithm will eventually terminate, giving us all the relevant categories and needed rules (we won't get them all, since the schemes could apply to non-relevant categories, but we don't want those in any case).

So here we go:

- starting rules: we search for all lexical items whose features ends with c. There are two here, giving two different relevant categories: $\epsilon :: \mathrm{c}$ and $\epsilon :: = \mathrm{a} + \mathrm{m}\ \mathrm{c}$. We have thus two rules:

  Start: *start* $\longrightarrow [\cdot \mathrm{c}]$

  Start: *start* $\longrightarrow [= \mathrm{a} + \mathrm{m} \cdot \mathrm{c}]$

  We have now two new relevant categories: $[\cdot \mathrm{c}]$ and $[= \mathrm{a} + \mathrm{m} \cdot \mathrm{c}]$. We will now write the rules with these on the left side of the arrow.

- $[\cdot \mathrm{c}]$ correspond to case 3. There is but one lexical item with features c, which is $\epsilon :: \mathrm{c}$, so we have a single rule:

  Lexicalize: $[\cdot \mathrm{c}] \longrightarrow \epsilon :: \mathrm{c}$

  No new relevant category is created, so we can move to the next one:

- $[= \mathrm{a} + \mathrm{m} \cdot \mathrm{c}]$ corresponds to the case 2b, so we can have two possibilities. Since there is no 'S', only the case 2(b)ii can apply. We must then look for lexical items whose last feature is $-\mathrm{m}$. There is but one (and thus only one corresponding relevant category), $a :: = \mathrm{b}\ \mathrm{a} - \mathrm{m}$. So we have one possible rule:

  Unmove-1: $[= \mathrm{a} + \mathrm{m} \cdot \mathrm{c}] \longrightarrow [= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{c}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$

  We have now a new relevant category, $[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{c}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$.

- $[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{c}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$ corresponds to case 2(a)i. For case 2(a)iA, we have to look for a lexical item whose last feature is a. Since there is no such item, we fall back to 2(a)iB. Here we have to look in 'S' for feature strings of type $\gamma\ \mathrm{a} \cdot \varphi$. There is only one, namely $= \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}$, so we have one rule:

Unmerge-3, simple: $[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{c}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}] \longrightarrow [\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{c}][= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$

We got here two more relevant categories, $[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{c}]$ and $[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$.

- $[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{c}]$ corresponds to case 3, and there is but one lexical item with the corresponding features, so we have one additional rule:

  Lexicalize:$[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{c}] \longrightarrow \epsilon :: = \mathrm{a} + \mathrm{m}\ \mathrm{c}$

- $[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$ corresponds to case 2(a)i. We first try case 2(a)iA. We look for lexical items with last feature b. There are two such items, namely $b :: \mathrm{b}$ and $b :: = \mathrm{a} + \mathrm{m}\ \mathrm{b}$. We then have two rules:

  Unmerge-1: $[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}] \longrightarrow [\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}][\cdot \mathrm{b}]$
  Unmerge-1: $[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}] \longrightarrow [\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}][= \mathrm{a} + \mathrm{m} \cdot \mathrm{b}]$

  Since 'S' is here empty, case 2(a)iB can't apply, and we move on to the three newly discovered relevant categories, $[\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}]$, $[\cdot \mathrm{b}]$ and $[= \mathrm{a} + \mathrm{m} \cdot \mathrm{b}]$.

- $[\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}]$ is ready to be lexicalized, there is still only one corresponding lexical item, so we get the rule:

  Lexicalize: $[\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}] \longrightarrow a :: = \mathrm{b}\ \mathrm{a} - \mathrm{m}$

- $[\cdot \mathrm{b}]$ is in the same case, we thus have:

  Lexicalize: $[\cdot \mathrm{b}] \longrightarrow b :: \mathrm{b}$

- $[= \mathrm{a} + \mathrm{m} \cdot \mathrm{b}]$ corresponds to the case 2(b)ii, with only one corresponding lexical item, thus the rule:

  Unmove-1: $[= \mathrm{a} + \mathrm{m} \cdot \mathrm{b}] \longrightarrow [= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{b}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$

- $[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{b}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$ corresponds to case 2(a)iB, and we have one rule:

  Unmerge-3, simple: $[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{b}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}] \longrightarrow [\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{b}][= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$

  Since $[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$ has already been treated, we can move to the last untreated relevant category:

- $[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{b}]$ is ready to be lexicalized:

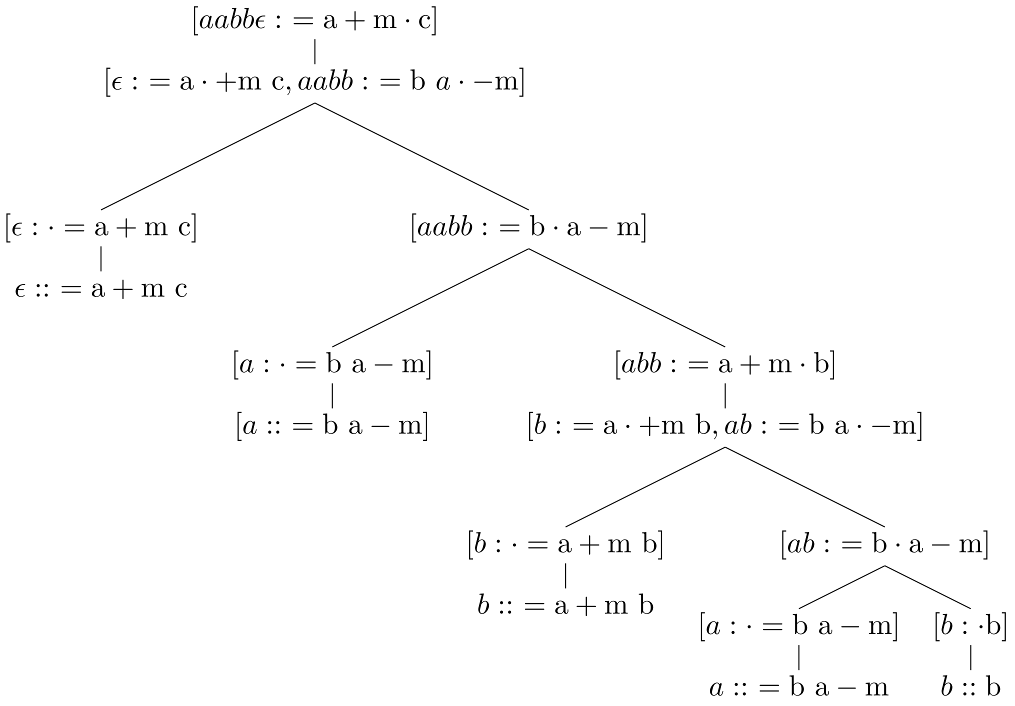
  Lexicalize: $[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{b}] \longrightarrow b :: = \mathrm{a} + \mathrm{m}\ \mathrm{b}$

We are now ready to give probabilities to these rules, conditioned by the left-hand side. The assignment here is quite easy : apart from the two cases where there are two possible rules (axiom choice and category and $[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$), the conditioned probability will be 1 (there is no choice). For the two other cases, we can assign any probability λ to one rule, and give the other a probability $1 - \lambda$. We can now give the following table:

$start$	$\longrightarrow [\cdot \mathrm{c}]$	Start	$\mathbb{P}(.) = \lambda$
$start$	$\longrightarrow [= \mathrm{a} + \mathrm{m} \cdot \mathrm{c}]$	Start	$\mathbb{P}(.) = 1 - \lambda$
$[\cdot \mathrm{c}]$	$\longrightarrow \epsilon :: \mathrm{c}$	Lexicalize	$\mathbb{P}(.) = 1$
$[= \mathrm{a} + \mathrm{m} \cdot \mathrm{c}]$	$\longrightarrow [= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{c}, \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$	Unmove-1	$\mathbb{P}(.) = 1$
$[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{c}, \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$	$\longrightarrow [\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{c}][= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$	Unmerge-3, simple	$\mathbb{P}(.) = 1$
$[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{c}]$	$\longrightarrow \epsilon :: = \mathrm{a} + \mathrm{m}\ \mathrm{c}$	Lexicalize	$\mathbb{P}(.) = 1$
$[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$	$\longrightarrow [\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}][\cdot \mathrm{b}]$	Unmerge-1, simple	$\mathbb{P}(.) = \mu$
$[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$	$\longrightarrow [\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}][= \mathrm{a} + \mathrm{m} \cdot \mathrm{b}]$	Unmerge-1, simple	$\mathbb{P}(.) = 1 - \mu$
$[\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}]$	$\longrightarrow a :: = \mathrm{b}\ \mathrm{a} - \mathrm{m}$	Lexicalize	$\mathbb{P}(.) = 1$
$[\cdot \mathrm{b}]$	$\longrightarrow b :: \mathrm{b}$	Lexicalize	$\mathbb{P}(.) = 1$
$[= \mathrm{a} + \mathrm{m} \cdot \mathrm{b}]$	$\longrightarrow [= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{b}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$	Unmove-1	$\mathbb{P}(.) = 1$
$[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{b}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$	$\longrightarrow [\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{b}][= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$	Unmerge-3, simple	$\mathbb{P}(.) = 1$
$[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{b}]$	$\longrightarrow b :: = \mathrm{a} + \mathrm{m}\ \mathrm{b}$	Lexicalize	$\mathbb{P}(.) = 1$

We will now end by giving the probability of a particular derivation tree:

(5)



All the rules here have probability 1, except the top one, the choice of the start rule $start \longrightarrow [aabb\epsilon : = \mathrm{a} + \mathrm{m} \cdot \mathrm{c}]$, which has probability $1 - \lambda$, the one from $[aabb : = \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$, which has probability $1 - \mu$, and the one from $[ab : = \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$, which has probability μ. So the complete tree has probability $\mu(1 - \mu)\lambda$, and, for example, the subtree headed by $[b : = \mathrm{a} \cdot +\mathrm{m}\ \mathrm{b}, ab : = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$ has probability μ.

2.4 The Cats and Mouses example

Let's now get back to our toy grammar (2) and see how it rewrites:

(6) $\bullet$*mouse* $:: \mathrm{n}$

$\bullet$*cat* $:: \mathrm{n}$

$\bullet$*the* $::= \mathrm{n\ d}$

$\bullet$*which* $::= \mathrm{n\ d} - \mathrm{wh}$

$\bullet$*ate* $::= \mathrm{d} = \mathrm{d\ c}$

$\bullet$*eat* $::= \mathrm{d} = \mathrm{d\ v}$

$\bullet$*did* $::= \mathrm{v} + \mathrm{wh\ c}$

$\bullet$*did* $::= \mathrm{v\ c}$

The rules are the following:

1	*start*	$\longrightarrow [= \mathrm{d} = \mathrm{d} \cdot \mathrm{c}]$	Start
2	*start*	$\longrightarrow [= \mathrm{v} + \mathrm{wh} \cdot \mathrm{c}]$	Start
3	*start*	$\longrightarrow [= \mathrm{v} \cdot \mathrm{c}]$	Start
4	$[= \mathrm{d} = \mathrm{d} \cdot \mathrm{c}]$	$\longrightarrow [= \mathrm{n} \cdot \mathrm{d}][= \mathrm{d} \cdot = \mathrm{d\ c}]$	Unmerge-2
5	$[= \mathrm{d} \cdot = \mathrm{d\ c}]$	$\longrightarrow [\cdot = \mathrm{d} = \mathrm{d\ c}][= \mathrm{n} \cdot \mathrm{d}]$	Unmerge-1
6	$[\cdot = \mathrm{d} = \mathrm{d\ c}]$	$\longrightarrow$ *ate* $:: = \mathrm{d} = \mathrm{d\ c}$	Lexicalize
7	$[= \mathrm{n} \cdot \mathrm{d}]$	$\longrightarrow [\cdot = \mathrm{n\ d}][\cdot \mathrm{n}]$	Unmerge-1
8	$[\cdot = \mathrm{n\ d}]$	$\longrightarrow$ *the* $:: = \mathrm{n\ d}$	Lexicalize
9	$[\cdot \mathrm{n}]$	$\longrightarrow$ *mouse* $::\ \mathrm{n}$	Lexicalize
10	$[\cdot \mathrm{n}]$	$\longrightarrow$ *cat* $::\ \mathrm{n}$	Lexicalize
11	$[= \mathrm{v} + \mathrm{wh} \cdot \mathrm{c}]$	$\longrightarrow [= \mathrm{v} \cdot +\mathrm{wh\ c}, = \mathrm{n\ d} \cdot -\mathrm{wh}]$	Unmove-1
12	$[= \mathrm{v} \cdot +\mathrm{wh\ c}, = \mathrm{n\ d} \cdot -\mathrm{wh}]$	$\longrightarrow [\cdot = \mathrm{v} + \mathrm{wh\ c}][= \mathrm{d} = \mathrm{d} \cdot \mathrm{v}, = \mathrm{n\ d} \cdot -\mathrm{wh}]$	Unmerge-1
13	$[\cdot = \mathrm{v} + \mathrm{wh\ c}]$	$\longrightarrow$ *did* $:: = \mathrm{v} + \mathrm{wh\ c}$	Lexicalize
14	$[= \mathrm{d} = \mathrm{d} \cdot \mathrm{v}, = \mathrm{n\ d} \cdot -\mathrm{wh}]$	$\longrightarrow [= \mathrm{n} \cdot \mathrm{d}][= \mathrm{d} \cdot = \mathrm{d\ v}, = \mathrm{n\ d} \cdot -\mathrm{wh}]$	Unmerge-2
15	$[= \mathrm{d} = \mathrm{d} \cdot \mathrm{v}, = \mathrm{n\ d} \cdot -\mathrm{wh}]$	$\longrightarrow [= \mathrm{n} \cdot \mathrm{d} - \mathrm{wh}][= \mathrm{d} \cdot = \mathrm{d\ v}]$	Unmerge-3, complex
16	$[= \mathrm{d} \cdot = \mathrm{d\ v}, = \mathrm{n\ d} \cdot -\mathrm{wh}]$	$\longrightarrow [\cdot = \mathrm{d} = \mathrm{d\ v}][= \mathrm{n} \cdot \mathrm{d} - \mathrm{wh}]$	Unmerge-3, simple
17	$[\cdot = \mathrm{d} = \mathrm{d\ v}]$	$\longrightarrow$ *eat* $:: = \mathrm{d} = \mathrm{d\ v}$	Lexicalize
18	$[= \mathrm{n} \cdot \mathrm{d} - \mathrm{wh}]$	$\longrightarrow [\cdot = \mathrm{n\ d} - \mathrm{wh}][\cdot \mathrm{n}]$	Unmerge-1
19	$[\cdot = \mathrm{n\ d} - \mathrm{wh}]$	$\longrightarrow$ *which* $:: = \mathrm{n\ d} - \mathrm{wh}$	Lexicalize
20	$[= \mathrm{d} \cdot = \mathrm{d\ v}]$	$\longrightarrow [\cdot = \mathrm{d} = \mathrm{d\ v}][= \mathrm{n} \cdot \mathrm{d}]$	Unmerge-1
21	$[= \mathrm{v} \cdot \mathrm{c}]$	$\longrightarrow [\cdot = \mathrm{v\ c}][= \mathrm{d} = \mathrm{d} \cdot \mathrm{v}]$	Unmerge-1
22	$[\cdot = \mathrm{v\ c}]$	$\longrightarrow$ *did* $:: = \mathrm{v\ c}$	Lexicalize
23	$[= \mathrm{d} = \mathrm{d} \cdot \mathrm{v}]$	$\longrightarrow [= \mathrm{n} \cdot \mathrm{d}][= \mathrm{d} \cdot = \mathrm{d\ v}]$	Unmerge-2

3 The probabilistic top-down parser

The parser will work on an ordered list of hypothesis, which he will expand in turn during the parse of the sentence. Before beginning presenting the algorithm, some definitions are needed:

3.1 Definitions

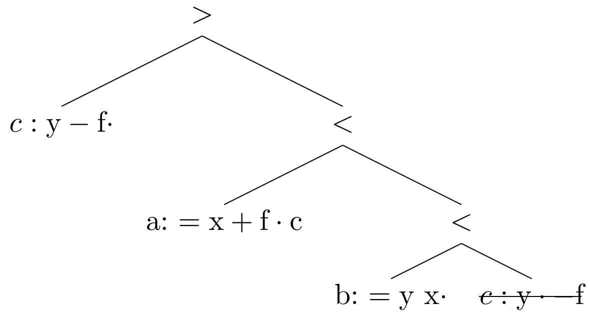
One difficulty in working with derivation trees instead of regular derived trees, is that the order of the words cannot be easily deduced (short of redoing the actual derivation). In order to keep track of the position of a category in the *derived tree* (so the parser may know in which order to expand the tree), we introduce *position indices*, which denotes positions in the derived tree from its root by a chain of digits (0 if going down left, 1 if going down right). From this perspective we can also define a *successor* operator on them, corresponding to a left-to-right sweep of the tree.

Consider the grammar given by the following lexical items:

(7) 1.a :: $=$ x $+$ f c
2.b :: $=$ y x
3.c :: y $-$ f

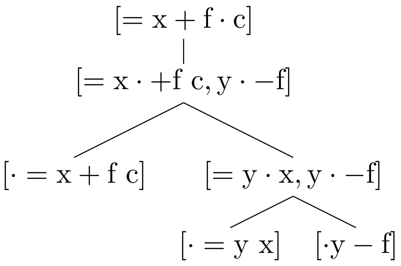
This grammar will generate the derived tree:

(8)



Corresponding to this tree is the derivation tree:

(9)



The parser should try to expand the nodes leading to the first leaf *in the derived tree* (8), but is actually building the *derivation tree* (9). As such, it should begin by expanding right-most nodes, then switch back to left-most ones when c is parsed to parse a, etc... Position indices showing in which position which category is can be computed online and incorporated to the derivation tree, for example 0/ for all categories corresponding to c, since its final position is just one branch down and left from the root. The parser will just have to expand the unexpanded nodes with lowest (i.e.

leftmost) position indices. In order to do this, it can keep track of a *pointer* telling up to which point nodes have been expanded, and expand the corresponding one. Then upgrading the pointer with the adequate notion of successor keeps the parser working. To formalize this:

Definition 3.1. *A* position index *is a element* $\pi \in \{0,1\}^* \cup \{-1\}$.
Its successor $s(\pi)$ *is defined to be:*

$$s(\pi) = \begin{cases} \alpha 1 & \textit{if } \pi = \alpha 0 \beta, \beta \in 1^* \\ -1 & \textit{if } \pi \in 1^* \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

Two positions indices π, π' correspond *if* $\pi' = \pi\beta, \beta \in 0^*$. *In this case, we say also that* π points to π'.

The notion of *correspondence* enables the parser to have some liberty in the pointer indicating the index to be expanded. Indeed, the parser will not try to expand the node with the index exactly equal to the pointer, but just corresponding to it, that is, equal to the pointer with as many 0s as possible following, or, in the derived tree, down the leftmost path from the node indicated by the pointer, which is what we would want: the first unexpanded node down the pointer.

Definition 3.2. *A* situated category *is a pair* $\alpha^n/[F^n]$*, where* α^n *is a sequence of n position indices and* F^n *is a sequence of n dotted feature strings (so that* $[F^n]$ *is a category). For readability, we will write* $\langle \alpha_1, \ldots, \alpha_n \rangle / [F_1, \ldots, F_n]$ *as* $[\alpha_1/F_1, \ldots, \alpha_n/F_n]$.

Definition 3.3. *A* hypothesis *is a 5-uple* (T, π, p, s, Δ) *where:*

- T *is a finite set of situated categories (the nodes of the partial derivation tree),*
- π *is a position index, the* pointer, *pointing to the next node to expand,*
- $p \in [0,1]$ *is the probability of the hypothesis,*
- s *is a dotted input string, and*
- Δ *is the sequence of rules used to obtain this hypothesis from the axiom start.*

The dotted input string s is the string of word of the phrase being parsed, with a dot indicating up to which point it has already been parsed (in fact, up to which point the words have been scanned). For example, if $s =$ "The cat has · eaten the mouse", this means that this hypothesis has already scanned (i.e., recognised a node for) the words "The", "cat" and "has", but not yet "eaten", "the" and "mouse".

3.2 Position indices and nodes

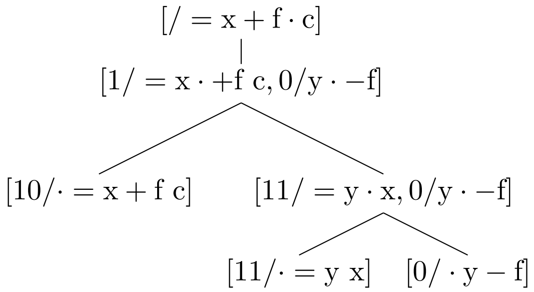
The parser will expand the hypothesis trees in a quite particular way, corresponding to a left-to-right reading of the output sentence. Since movement is possible in MG, the parser will have to keep track of the 'position' of the different elements, to only expand the leafs corresponding to the currently parsed word. This is the role of the position indices.

A position index different from -1 will represent a particular subtree in the final *derived* tree, where the traces of the moved nodes are deleted (moving up its sister to the position of its mother). The position index $\alpha_0 \dots \alpha_k$ corresponds to the subtree dominated by the node obtained by going down in the tree from its root, left if $\alpha_0 = 0$, right otherwise, then again left if $\alpha_1 = 0$, right otherwise, etc...

Back to our toy grammar (7):

1. $a :: = \mathrm{x} + \mathrm{f}\ \mathrm{c}$
2. $b :: = \mathrm{y}\ \mathrm{x}$
3. $c :: \mathrm{y} - \mathrm{f}$

The derivation tree with indexed relevant features corresponding to (8):



The indexed relevant category at the root of the derivation tree has a empty position string since it represent the derived tree itself, and in $[11/ = \mathrm{y} \cdot \mathrm{x}, 0/\mathrm{y} \cdot -\mathrm{f}]$ for example, we have $11/ = \mathrm{y} \cdot \mathrm{x}$ because this relevant category represent the tree under the node obtained if you go right (1), then right again (11) from the root node of the derived tree (without the moving categories, since they will move so won't end up at the same place). We have similarly $0/\mathrm{y} \cdot -\mathrm{f}$ because the subtree described by $\mathrm{y} \cdot -\mathrm{f}$ ends up as the left (0) daughter of the root of the derived tree.

The assignment of these position strings is given by the inference rules, which will be discussed later.

3.3 Axiom

The axiom of the parser are exactly the same as the axiom for the LCFRS corresponding to our MG discussed in 2, plus an empty position string (it represents the whole derived tree...). Its probability will be of course 1, and the pointer will be set as ϵ. So, if the phrase to be parsed is ω, we have a parser axiom $(\epsilon/\mathit{start}, \epsilon, 1, \cdot\omega, \langle\ \rangle)$.

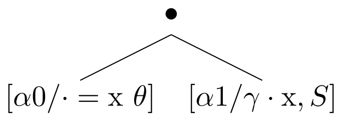
3.4 Inference rules

We here have exactly the same inference rules as before, exept that these will assign position strings too. So here they are:

1. Start rules: for every lexical item $\gamma :: \delta\ \mathrm{c}$,
   Start: $\epsilon/\mathit{start} \longrightarrow [\epsilon/\delta \cdot \mathrm{c}]$

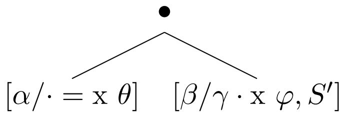
2. Un-merge rules: the left-hand category is of the form $[\alpha/\delta = \mathrm{x}\cdot\theta, S]$
   (a) Cases where the selector was a simple tree ($\delta = \epsilon$):
      i. For any lexical item of feature string γ x,
      Unmerge-1:

      $[\alpha/ = \mathrm{x}\cdot\theta, S] \longrightarrow$


●
$[\alpha 0/\cdot = \mathrm{x}\ \theta]$
$[\alpha 1/\gamma\cdot \mathrm{x}, S]$

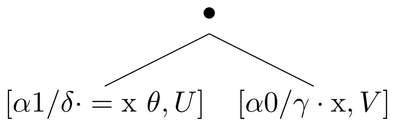
      t is here s if $\gamma = \emptyset$ (and thus $S = \emptyset$ too), and c otherwise.
      ii. For any element $(\gamma\ \mathrm{x}\cdot\varphi) \in S$, with $S' = S - (\gamma\ \mathrm{x}\cdot\varphi)$,
      Unmerge-3, simple:

      $[\alpha/ = \mathrm{x}\cdot\theta, \beta/\gamma\ \mathrm{x}\cdot\varphi, S'] \longrightarrow$


●
$[\alpha/\cdot = \mathrm{x}\ \theta]$
$[\beta/\gamma\cdot \mathrm{x}\ \varphi, S']$

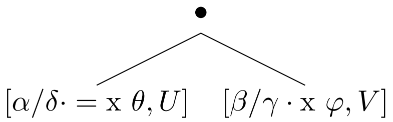
      t is here s if $\gamma = \emptyset$ (and thus $S' = \emptyset$ too), and c otherwise. It should be noted that necessarily, $\varphi \neq \emptyset$.
   (b) Cases where the selector was a complex tree:
      i. For any decomposition $S = U \sqcup V$, and any lexical item of feature string γ x,
      Unmerge-2:

      $[\alpha/\delta = \mathrm{x}\cdot\theta, S] \longrightarrow$


●
$[\alpha 1/\delta\cdot = \mathrm{x}\ \theta, U]$
$[\alpha 0/\gamma\cdot \mathrm{x}, V]$

      t is, as always, s if $\gamma = \emptyset$ (and thus V has to be empty too), and c otherwise.
      ii. For any element $(\gamma\ \mathrm{x}\cdot\varphi) \in S$, and any decomposition $S = U \sqcup V \sqcup (\gamma\ \mathrm{x}\cdot\varphi)$,
      Unmerge-3, complex:

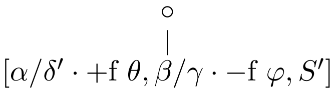
      $[\alpha/\delta = \mathrm{x}\cdot\theta, \beta/\gamma\ \mathrm{x}\cdot\varphi, S'] \longrightarrow$


●
$[\alpha/\delta\cdot = \mathrm{x}\ \theta, U]$
$[\beta/\gamma\cdot \mathrm{x}\ \varphi, V]$

      t is still s if $\gamma = \emptyset$ (and thus V has to be empty too), and c otherwise. As in 2(a)iB, φ has to be non empty.

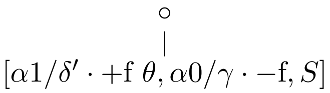
3. Un-move rules: the left-hand category is of the form $[\delta + \mathrm{f}\cdot\theta, S]$
   (a) For any $(\gamma - \mathrm{f}\cdot\varphi) \in S$ (necessarily unique by the Shortest Movement Constraint), with $S' = S - (\gamma - \mathrm{f}\cdot\varphi)$,
   Unmove-2:

   $[\alpha/\delta' + \mathrm{f}\cdot\theta, \beta/\gamma - \mathrm{f}\cdot\varphi, S'] \longrightarrow$


○
$[\alpha/\delta'\cdot +\mathrm{f}\ \theta, \beta/\gamma\cdot -\mathrm{f}\ \varphi, S']$

   (b) If there is no $(\gamma - \mathrm{f}\cdot\varphi) \in S$, then for any lexical item of feature string $\gamma - \mathrm{f}$,
   Unmove-1:

   $[\alpha/\delta' + \mathrm{f}\cdot\theta, S] \longrightarrow$


○
$[\alpha 1/\delta'\cdot +\mathrm{f}\ \theta, \alpha 0/\gamma\cdot -\mathrm{f}, S]$

There is no lexicalise rule, since it will in fact be replaced by a '*scan* rule', checking if the feature string of the word currently parsed corresponds to the current feature string.

3.5 Top-down parser

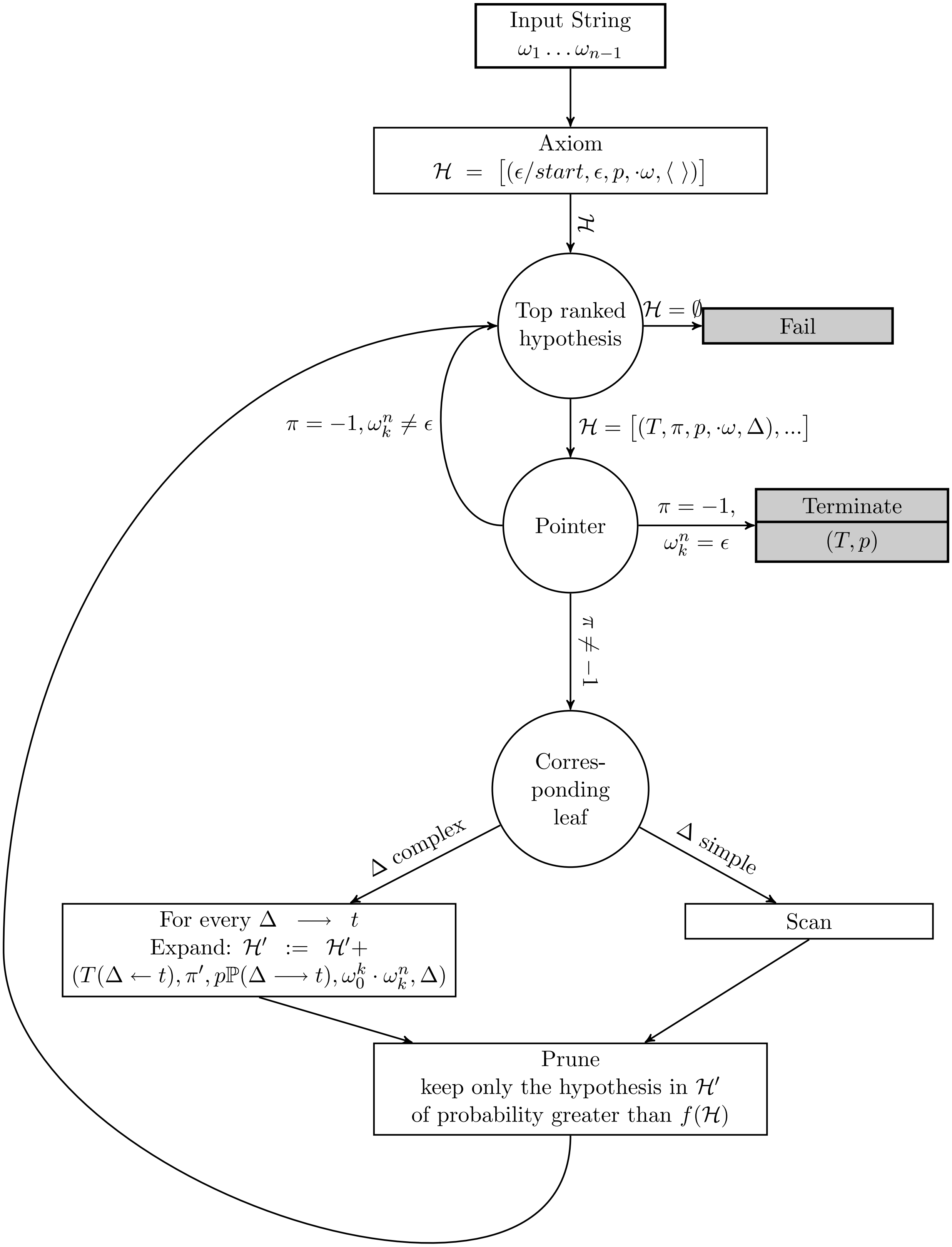
The parser takes an *input string* $\omega = \omega_0 \dots \omega_{n-1}$, a minimalist grammar $\mathcal{G}$ rewrited into a LCFRS $\mathcal{S}$ and a *beam function* f, setting a threshold to the probability of the selected hypothesis. It will work on a *priority queue* of hypothesis $\mathcal{H}$. The function f can be very general, here we will consider that its argument is the priority queue $\mathcal{H}$. The parser works as following:

1. **Beginning:** The parser start with the queue of hypothesis consisting of the axiom $(\epsilon/start, \epsilon, 1, \cdot\omega, \langle\, \rangle)$ of the grammar.
2. **Expanding:** At each step, the parser will:
   - take the top-ranked hypothesis $(T, \pi, p, \omega_0^k \cdot \omega_k^n, \Delta)$ (i.e. the hypothesis with greatest p) in the priority queue,
   - check the corresponding position string pointer. If $\pi = -1$, and the parsing dot in $\omega_0^k \cdot \omega_k^n$ is at the far right (i.e. $k = n$), then the parser terminates and returns the sequence of rules Δ. If the phrase is not completely parsed ($\pi = -1$ but $k < n$), the hypothesis is deleted and the parser moves to the next one. If $\pi \neq -1$, the parser moves to the next step, and tries to:
   - find the leaf of T, C, in which is the position string α corresponding to the pointer π. If $\alpha \neq \pi$, π is set to α.
   - expand C. For this, we have two possibilities:
     (a) **Expand:** If C is a complex situated category, the parser will delete the current hypothesis $(T, p, \pi, \omega_0^k \cdot \omega_k^n, \Delta)$ and add to the priority queue, for all possible inference rules $C \longrightarrow t$, a new hypothesis $(T', \pi', p\mathbb{P}(C \longrightarrow t), \omega_0^k \cdot \omega_k^n, \Delta@C \longrightarrow t)$, such that T' is T where the node C has been replaced by t, and π' is either $\pi 0$ if the rule did change the value of the position string corresponding to π (i.e. if the rule was Unmerge-1, Unmerge-2 and Unmove-1, and the first element of C had α for position string), and π in the other cases. @ is the concatenation operator.
     (b) **Scan:** If C is a simple indexed category, say $C = [\alpha/\cdot\delta]$, then the parser will delete the current hypothesis $(T, \pi, p, \omega_0^k \cdot \omega_k^n, \Delta)$, and try to lexicalize C. It will do two things:
       i. **Scan, ϵ:** If there is a rule $[\cdot\delta] \longrightarrow \epsilon :: \delta$, then a new hypothesis $(T', s(\pi), p\mathbb{P}([\cdot\delta] \longrightarrow \epsilon :: \delta), \omega_0^k \cdot \omega_k^n, \Delta@[\cdot\delta] \longrightarrow \epsilon :: \delta)$ is added to the priority queue, where T' is T where the leaf C was replaced by $\epsilon :: \delta$.
       ii. **Scan, $\not\epsilon$:** If $\omega_k :: \delta$ is in the grammar, then a new hypothesis $(T', s(\pi), p\mathbb{P}([\cdot\delta] \longrightarrow \omega_k :: \delta), \omega_0^k\omega_k \cdot \omega_{k+1}^n, \Delta@[\cdot\delta] \longrightarrow \omega_k :: \delta)$ is added to the priority queue, where T' is T where the leaf C was replaced by $\omega_k :: \delta$.

       If these two steps fail, then no hypothesis is added to the priority queue.

     The new hypothesis are inserted in the priority queue at their 'right place', i.e. after all hypothesis of higher probability.
   - **Prune:** The parser deletes all hypothesis of the priority queue whose probability is lower than $f(\mathcal{H})$.

   If $\mathcal{H}$ is empty, then the parse failed and the sentence is judged ungrammatical.

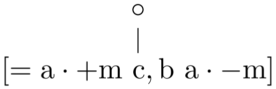
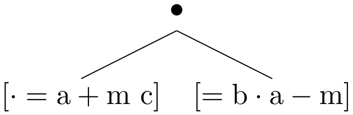
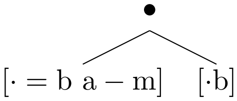
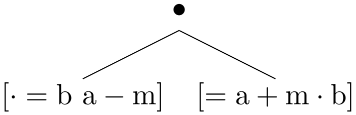
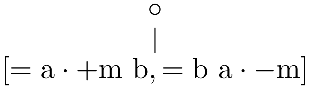
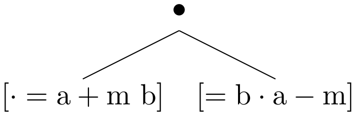

Input String
$\omega_1 \dots \omega_{n-1}$
Axiom
$\mathcal{H} = \big[(\epsilon/start, \epsilon, p, \cdot\omega, \langle\ \rangle)\big]$
$\mathcal{H}$
Top ranked hypothesis
$\mathcal{H} = \emptyset$
Fail
$\pi = -1, \omega_k^n \neq \epsilon$
$\mathcal{H} = \big[(T, \pi, p, \cdot\omega, \Delta), ...\big]$
Pointer
$\pi = -1,$
$\omega_k^n = \epsilon$
Terminate
(T, p)
$\pi \neq -1$
Corres-ponding leaf
Δ complex
Δ simple
For every $\Delta \longrightarrow t$
Expand: $\mathcal{H}' := \mathcal{H}' +$
$(T(\Delta \leftarrow t), \pi', p\mathbb{P}(\Delta \longrightarrow t), \omega_0^k \cdot \omega_k^n, \Delta)$
Scan
Prune
keep only the hypothesis in $\mathcal{H}'$
of probability greater than $f(\mathcal{H})$

3.6 Example

Here we will present a small example of the parsing of a particular sentence of the grammar we presented in (4), consisting of the the lexical items:

1. $\epsilon :: \mathrm{c}$
2. $\epsilon :: = \mathrm{a} + \mathrm{m}\ \mathrm{c}$
3. $a :: = \mathrm{b}\ \mathrm{a} - \mathrm{m}$
4. $b :: \mathrm{b}$
5. $b :: = \mathrm{a} + \mathrm{m}\ \mathrm{b}$

The corresponding LCFRS was consisting of these rules:

S1	*start*	$\longrightarrow$	$[\cdot \mathrm{c}]$	Start	$\mathbb{P}(.) = .7$
S2	*start*	$\longrightarrow$	$[= \mathrm{a} + \mathrm{m} \cdot \mathrm{c}]$	Start	$\mathbb{P}(.) = .3$
L1	$[\cdot \mathrm{c}]$	$\longrightarrow$	$\epsilon :: \mathrm{c}$	Lexicalize	$\mathbb{P}(.) = 1$
Mv1	$[= \mathrm{a} + \mathrm{m} \cdot \mathrm{c}]$	$\longrightarrow$	 $\circ$ $[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{c}, \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$	Unmove-1	$\mathbb{P}(.) = 1$
Mg1	$[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{c}, \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$	$\longrightarrow$	 $\bullet$ $[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{c}]$ $[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$	Unmerge-3, simple	$\mathbb{P}(.) = 1$
L2	$[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{c}]$	$\longrightarrow$	$\epsilon :: = \mathrm{a} + \mathrm{m}\ \mathrm{c}$	Lexicalize	$\mathbb{P}(.) = 1$
Mg2	$[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$	$\longrightarrow$	 $\bullet$ $[\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}]$ $[\cdot \mathrm{b}]$	Unmerge-1, simple	$\mathbb{P}(.) = .4$
Mg3	$[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$	$\longrightarrow$	 $\bullet$ $[\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}]$ $[= \mathrm{a} + \mathrm{m} \cdot \mathrm{b}]$	Unmerge-1, simple	$\mathbb{P}(.) = .6$
L3	$[\cdot = \mathrm{b}\ \mathrm{a} - \mathrm{m}]$	$\longrightarrow$	$a :: = \mathrm{b}\ \mathrm{a} - \mathrm{m}$	Lexicalize	$\mathbb{P}(.) = 1$
L4	$[\cdot \mathrm{b}]$	$\longrightarrow$	$b :: \mathrm{b}$	Lexicalize	$\mathbb{P}(.) = 1$
Mv2	$[= \mathrm{a} + \mathrm{m} \cdot \mathrm{b}]$	$\longrightarrow$	 $\circ$ $[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{b}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$	Unmove-1	$\mathbb{P}(.) = 1$
Mg4	$[= \mathrm{a} \cdot +\mathrm{m}\ \mathrm{b}, = \mathrm{b}\ \mathrm{a} \cdot -\mathrm{m}]$	$\longrightarrow$	 $\bullet$ $[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{b}]$ $[= \mathrm{b} \cdot \mathrm{a} - \mathrm{m}]$	Unmerge-3, simple	$\mathbb{P}(.) = 1$
L5	$[\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{b}]$	$\longrightarrow$	$b :: = \mathrm{a} + \mathrm{m}\ \mathrm{b}$	Lexicalize	$\mathbb{P}(.) = 1$

Here we took $\lambda = .7$ and $\mu = .4$.

We will now try to parse the string *aabb*, which is generated by the grammar (the ϵ is of course omitted).

- The parser begins with his stack consisting of the *axiom* of the grammar:

$$\mathcal{H} = ((\epsilon/start, \epsilon, 1, \cdot aabb, \langle\ \rangle))$$

- The parsers takes the top-ranked hypothesis, $(\epsilon/start, \epsilon, 1, \cdot aabb, \langle\ \rangle)$, its pointer (null), the corresponding leaf, *start*, and tries to expand it. There are two possibilities, which are added to the hypothesis queue, ordered by decreasing possibilities:

$$\mathcal{H} = ((\mathrm{[}\epsilon/\cdot\ \mathrm{c]}, \epsilon, .7, \cdot aabb, \langle S1\rangle), (\mathrm{[}/ = \mathrm{a} + \mathrm{m}\cdot \mathrm{c]}, \epsilon, .3, \cdot aabb, \langle S2\rangle))$$

- The parser now takes the top-ranked hypothesis, $(\mathrm{[}\epsilon/\cdot\ \mathrm{c]}, \epsilon, .7, \cdot aabb, \langle S1\rangle)$, its pointer (null), the corresponding leaf, [/ · c], and try to scan it since it is a simple category. There is only one rule whose left-size is [/ · c], L1, with probability 1. The corresponding word is empty, so the scan succeeds, the pointer is increased to -1 and the new hypothesis is added to the queue in place of the old one:

$$\mathcal{H} = ((\mathrm{[}\epsilon/\epsilon :: \mathrm{c]}, -1, .7, \cdot aabb, \langle S1, L1\rangle), (\mathrm{[}/ = \mathrm{a} + \mathrm{m}\cdot \mathrm{c]}, \epsilon, .3, \cdot aabb, \langle S2\rangle))$$

- The parser takes once more the top-ranked analysis, $(\mathrm{[}\epsilon/\epsilon :: \mathrm{c]}, -1, .7, \cdot aabb, \langle S1, L1\rangle)$. Its pointer is -1, so the parser checks if the parse is indeed over. No, since the string left of the dot is non-empty. The current hypothesis is now deleted, and the new queue is fed to the parser:

$$\mathcal{H} = ((\mathrm{[}/ = \mathrm{a} + \mathrm{m}\cdot \mathrm{c]}, \epsilon, .3, \cdot aabb, \langle S2\rangle))$$

- The top-ranked analysis is now the only one in the queue, $(\cdot aabb, \mathrm{[}/ = \mathrm{a} + \mathrm{m}\cdot\mathrm{c]}, .3, \ )$. Its pointer is null, the corresponding leaf is [/ = a + m · c], which the parser will try to expand. There is only one rule, Mv1:$[= \mathrm{a} + \mathrm{m}\cdot \mathrm{c}] \longrightarrow \begin{array}{c}\circ\\ | \\ [= \mathrm{a}\cdot +\mathrm{m}\ \mathrm{c}, \mathrm{b}\ \mathrm{a}\cdot -\mathrm{m}]\end{array}$, so the hypothesis is replaced by the new one:

$$\mathcal{H} = \left((\begin{array}{c}\circ\\ | \\ [1/= \mathrm{a}\cdot +\mathrm{m}\ \mathrm{c}, 0/\mathrm{b}\ \mathrm{a}\cdot -\mathrm{m}]\end{array}, 0, .3, \cdot aabb, \langle S2, Mv1\rangle) \right)$$

  (note that the pointer was modified since the position vector of the expanded element was modified by the rule)

- The parser goes on, giving the new queue:

$$\mathcal{H} = \left((\begin{array}{c}\circ\\ | \\ \bullet \\ \diagup \qquad \diagdown \\ [1/\cdot = \mathrm{a} + \mathrm{m}\ \mathrm{c}] \quad [0/ = \mathrm{b}\cdot \mathrm{a} - \mathrm{m}]\end{array}, 0, .3, \cdot aabb, \langle S2, Mv1, Mg1\rangle) \right)$$

- And so forth...

4 Proofs of soundness and completeness

4.1 Soundess of the pointer

We will here demonstrate that the parser, at each step, will indeed find the correct leaf to expand with the current pointer.

First we modify a little, for convenience of the proof, our definition of a position string:

Definition 4.1. *A* position index *(and a* pointer*) is a dotted, almost null binary sequence $\alpha \cdot \beta, \alpha\beta \in \{0,1\}^k\bar{0}$ for some k.*

A position index and a pointer correspond *if their undotted sequences are the same.*

The set of all undotted position indexes is naturally ordered by the lexicographic order.

Note: This definition is consistent with the one I proposed in the precedent section, the dot being the place where the 'new' position indexes are to be truncated to obtain the 'old' ones.

Proposition 4.1. *There is a one-to-one application from the set of all position indexes $\mathcal{I}$ to the set $\mathcal{ST}$ of all subtrees of the infinite complete binary tree $\mathcal{T}$. The head of the subtrees corresponding to the positions indexes of the type $\alpha_1 \ldots \alpha_n \cdot \bar{0}$ are exactly the nodes of depth n of the tree. We thus have a notion of domination on the position indexes, corresponding to the notion of domination in the tree (by convention, a node dominates itself). A position index $\alpha \cdot \bar{0}$ dominate another position index $\alpha' \cdot \bar{0}$ if α is a prefix of α'.*

Proof. Let $\phi : \begin{array}{ccc} \mathcal{I} & \longrightarrow & \mathcal{ST} \\ \alpha_1 \ldots \alpha_n \cdot \bar{0} & \longrightarrow & T \end{array}$ where T is the subtree headed by the node obtained by, starting from the root of $\mathcal{T}$, for each α_i, going left if $\alpha_i = 0$ and right otherwise. This application has clearly all the above properties. □

Lemma 4.1. *For every cut C of position indexes,*

i. if $\beta \cdot \bar{0}$ is in the cut C, then there is no $\beta\gamma \cdot \bar{0}$ in C, for every $\gamma \in \{0,1\}^+$.

ii. If $\beta 0 \gamma \cdot \bar{0}$ is in the cut C, then there is a unique $k \in \mathbb{N}$ such that $\beta 1 0^k \cdot \bar{0}$ is in the cut.

iii. The lexicographic order can be extended to the dotted *elements of C.*

Proof. Let n be the depth of the cut.

i. Suppose that we have $\beta \cdot \bar{0}$ and $\beta\gamma \cdot \bar{0}$ in the cut, for some $\gamma \in \{0,1\}^+$. Then $\beta\gamma 0^n \cdot \bar{0}$ is dominated by both $\beta \cdot \bar{0}$ and $\beta\gamma \cdot \bar{0}$, which is a contradiction.

ii. Since C is a cut, $\beta 1 0^n \cdot \bar{0}$ must be dominated by a unique element of C, say $\delta \cdot \bar{0}$. δ is then a prefix of $\beta 1 0^n$. But δ cannot be a prefix of $\beta 0 \gamma$, since otherwise $\delta \cdot \bar{0}$ would dominate $\beta 0 \gamma \cdot \bar{0}$, which is already dominated by itself. So δ must be of the form $\beta 1 0^k, k < n$. The unicity follows from i.

iii. This follows directly from i.

□

We can now prove that the parser will never have a pointer problem:

Theorem 4.1. *At each point of the parse, the position indexes of (all) the hypothesis form a cut, the already scanned position indexes form a prefix set of all the position indexes (for the lexicographic order), and the pointer correspond to the smallest unscanned position index (which exists since the set is finite), or is possibly -1 if there is none.*

Proof. We'll prove this result by recurrence on the number of steps done by the parser.

- At the beginning, the set of the position indexes is reduced to $\{\cdot\bar{0}\}$, and the pointer is $\cdot\bar{0}$, so the result is trivially true.
- Suppose that at step n, the position indexes of (all) the hypothesis form a cut, the already scanned position indexes form a prefix set of all the position indexes (for the lexicographic order), and the pointer correspond to the smallest scanned position index (which exists since the set is finite), or is possibly -1 if there is none. Let $\alpha \cdot \bar{0}$ be the position index corresponding to the pointer (unique by hypothesis), and $[\beta \cdot \bar{0}/\Delta, \ldots, \alpha \cdot \bar{0}/\Delta', \ldots]$ the leaf to be expanded. By hypothesis, $\alpha \cdot \bar{0} \leq \beta \cdot \bar{0}$. Let $(\delta_1, \ldots, \delta_k // \alpha \cdot \bar{0}, \ldots, \beta \cdot \bar{0}, \ldots)$ the positions indexes, lexicographically ordered, the scanned ones left of $//$. By hypothesis, this is a cut. The state of the parser at step $n+1$ can be obtained by four different cases:

  1. if the position index corresponding to the pointer, $\alpha \cdot \bar{0} (< \beta \cdot \bar{0})$, is not modified by the rules, two cases:

     (a) if the main position index of the leaf being expanded, $\beta \cdot \bar{0}$, is not modified (i.e. during Un-merge-3 or Un-move-2), no position indexes are modified, nor the pointer, so the result still holds.

     (b) if the main position index of the leaf being expanded, $\beta \cdot \bar{0}$, is modified (i.e. during Un-merge-1,-2 or Un-move-1), the new position indexes are (lexicographically ordered, the already scanned ones left of the $//$)
     $(\delta_1, \ldots, \delta_k // \alpha \cdot \bar{0}, \ldots, \beta 0 \cdot \bar{0}, \beta 1 \cdot \bar{0}, \ldots)$.
     This is trivially still a cut, the pointer is still $\alpha \cdot \bar{0}$, corresponding to the smallest unscanned position indexes $\alpha \cdot \bar{0}$.

  2. if the position index corresponding to the pointer, $\alpha \cdot \bar{0} (= \beta \cdot \bar{0})$, is modified by the rules, the new position indexes are
     $(\delta_1, \ldots, \delta_k // \alpha 0 \cdot \bar{0}, \alpha 1 \cdot \bar{0}, \ldots)$.
     This is still trivially a cut, and the new pointer is $\alpha 0 \cdot \bar{0}$, corresponding to the smallest unscanned position index $\alpha 0 \cdot \bar{0}$.

  3. if the parser *scans* the leaf (and then $\alpha \cdot \bar{0} = \beta \cdot \bar{0}$):
     If $\alpha \in 1^*$, the pointer is set to -1, there cannot be a greater item in the cut than $\alpha \in 1^*$ (since it would then have to have α as a prefix, which is impossible by lemma 4.1), so the result is true.
     Let's then write $\alpha = \gamma 0 1^m$. The pointer is set to $\gamma 1 \cdot \bar{0}$. The new position indexes are
     $(\delta_1, \ldots, \delta_k, \gamma 0 1^m \cdot \bar{0} // \zeta \cdot \bar{0}, \ldots)$.
     Since the only thing that changed here is the position of the $//$, this is still a cut. By lemma 4.1, there exists a unique $\xi = \gamma 1 0^r$ in the cut, since $\alpha = \gamma 0 1^m$ was in it. This being the smallest position index greater than $\alpha = \gamma 0 1^m$, we have $\zeta = \xi = \gamma 1 0^r$, which is corresponding to the new pointer. This ends the demonstration.

□

4.2 Completeness

Here we demonstrate the completeness of the parser, without the pruning step, i.e. that if the string to be parsed can indeed be generated by the grammar, then the parser will eventually parse it.

Lemma 4.2. *Let $(a_n)_{n\in\mathbb{N}} \in (\Sigma^*)^{\mathbb{N}}$ a infinite sequence of finite distinct strings over a finite alphabet Σ. Suppose that the following hold:*

$$\forall n \in \mathbb{N}, \text{ all prefixes of } a_n \text{ are in } \{a_k, k \leq n\}. \tag{1}$$

Then there exists an infinite sequence $(x_k)_{k\in\mathbb{N}} \in \Sigma^{\mathbb{N}}$ such that $(x_0 \dots x_k)_{k\in\mathbb{N}}$ is a subsequence of $(a_n)_{n\in\mathbb{N}}$.

Proof. Let's build this sequence recursively:

- Among all elements of Σ, there is an element which is prefix of infinitely many elements of $(a_n)_{n\in\mathbb{N}}$. Let's call it x_0. By hypothesis, $x_0 \in (a_n)_{n\in\mathbb{N}}$.
- Suppose we have $x_0, \dots, x_N$ such that $(x_0 \dots x_k)_{k\in[\![0,N]\!]}$ is a finite subsequence of $(a_n)_{n\in\mathbb{N}}$. Without loss of generality, we can restrict $(a_n)_{n\in\mathbb{N}}$ to its subsequence composed of the elements $(x_0 \dots x_k)$, $k \in [\![0, N]\!]$ and all elements of $(a_n)_{n\in\mathbb{N}}$ with prefix $x_0 \dots x_N$. This new sequence is still infinite, and has the prefix property 1.

  Among all elements of Σ, there is an element x such that $x_0 \dots x_N x$ is prefix of infinitely many elements of $(a_n)_{n\in\mathbb{N}}$. Let $x = x_{N+1}$. By hypothesis, $x_{N+1} \in (a_n)_{n\in\mathbb{N}}$.
- $(x_k)_{k\in\mathbb{N}}$ has the property we seek.

□

Theorem 4.2. *For all $p \in (0, 1]$, if there is no looping chain of rules of probability 1 in the grammar, there are finitely many partial derivation trees (PDT) of probability $\geq p$.*

Proof. A partial derivation tree is exactly defined by the string of rules deriving it. So a PDT will here be seen, when convenient, as a string of rules.

Suppose there were infinitely many PDT of probability $\geq p$. Then we have a sequence of strings of rules as defined by lemma 4.2. The lemma then gives us a sequence $A_0, \dots, A_n, \dots$ of rules such that $\forall n\ A_0 \dots A_n$ defines a correct partial derivation tree (since $A_0 \dots A_n$ is in the initial sequence, composed of correct PDTs).

To this sequence of rules corresponds an infinite sequence of growing PDT, all of which have probability $\geq p$. Since the sequence is growing and infinite, there is an infinite path in the limit of the trees, given by the sequence of rules $(A_{\varphi(n)})_{n\in\mathbb{N}}$.

Or, differently put, there is a sequence of rules $(A_{\varphi(n)})_{n\in\mathbb{N}}$, such that for all $n \geq 1$, the left side of A_n is in the right side of A_{n-1}.

Then there is a finite sequence $A_{\varphi(k)}, \dots, A_{\varphi(k')}$ such that

$$(A_{\varphi(n)})_{n\in\mathbb{N}} = A_{\varphi(0)}, A_{\varphi(1)}, \dots, A_{\varphi(k-1)}, (A_{\varphi(k)}, \dots, A_{\varphi(k')})^{\mathbb{N}}. \tag{2}$$

Indeed, let $\mathcal{A}$ be the finite state automaton with:

- states $A_{\varphi(n)}, n \in \mathbb{N}$ and END,
- starting state $A_{\varphi(0)}$, ending state END,

- transitions rules $A_{\varphi(n)} \longrightarrow A_{\varphi(n+1)}$ and $A_{\varphi(n)} \longrightarrow END$ for all n.

This automaton generates exactly all prefixes of $(A_{\varphi(n)})_{n\in\mathbb{N}}$. By the pumping lemma, there is a string $A_{\varphi(0)}A_{\varphi(1)}\dots A_{\varphi(N)}$ and two integers k, k' such that $A_{\varphi(0)}A_{\varphi(1)}\dots A_{\varphi(k-1)}(A_{\varphi(k)}\dots A_{\varphi(k')})^* A_{\varphi(k'+1)}\dots A_{\varphi(N)}$ is in the language recognised by $\mathcal{A}$, that is, prefixes of $(A_{\varphi(n)})_{n\in\mathbb{N}}$. This is exactly 2.

Let $p' = \prod_k^{k'} \mathbb{P}(A_{\varphi(i)})$. Then for all $n \in \mathbb{N}$, p'^n is an upper bound of the probability of some PDT of probability $\geq p$ (take any PDT where $(A_{\varphi(k)}, \dots, A_{\varphi(k')})^n$ is in the sequence of its rules). Thus $p'^n \geq p > 0\ \forall n$, which is impossible unless $p' = 1$. This contradicts the hypothesis that no looping chain of rules in the grammar has probability 1. □

Corollary 4.1. *If there is no looping chain of rules of probability* 1 *in the grammar, the parser is complete.*

Proof. If there is no looping chain of rules of probability 1, then Theorem 4.2 holds.

Let $A_0, \dots, A_n$ be the sequence of rules giving the parse of the parsed string. We'll show that for all $k \in [\![0, n]\!]$, the parse will have after finitely many steps $A_0 \dots A_k$ as its top-ranked hypothesis.

Proof. Let $p_k = \prod_{i=0}^k \mathbb{P}(A_i)$. Let's prove the result by recursion on k:

- for $k = 0$, A_0 is an axiom so is in the priority queue from the beginning of the parse. By theorem 4.2, there are finitely many PDTs of probability greater than p_0, say N. Since the parser will have each of them at most once as its top-ranked hypothesis, A_0 will be the top-ranked hypothesis before step $N + 1$.

- suppose that after M steps, $A_0 \dots A_k$ is the top-ranked hypothesis of the parser. Then at the $(M + 1)^{th}$ step, the parser will expand $A_0 \dots A_k$, and put (among others) $A_0 \dots A_k A_{k+1}$ in the priority queue. Since, by theorem 4.2, there are finitely many PDTs of probability greater than p_{k+1}, say N, and the parser will have each of them at most once as its top-ranked hypothesis, $A_0 \dots A_k A_{k+1}$ will be the top-ranked hypothesis before step $M + 1 + N + 1$.

  This completes the recursion.

□

The result follows from the case $k = n$. □

5 Conditioning the rules with the CTW algorithm

In this section we present a way to improve the performances of the parser, using the Context Tree Weighting (CTW) algorithm, whose description and properties may be found in [WST95] and an implementation in [FP]. The algorithm is originally intended to be used in data compression, but its construction of context trees allows us to use it in our parser.

The CTW algorithm uses a double mixture of context trees, and its force resides in the fact that conditional probabilities may be computed recursively, allowing for a great decrease in compuation time. Its idea is to mix together the Krichevski-Trofimov estimators for all possible context trees of depth less than a certain M, allowing for a near-optimal coding (and as such, estimate of conditional probabilities, in the sense of the Kullback-Leibler divergence).

5.1 Quick overview of the CTW

The setting is the following : consider a stationary ergodic source of unknown law $\mathbb{P}$. We want to estimate $\mathbb{P}$ by $\hat{\mathbb{P}}$, which will be a mixture of context tree laws.

5.1.1 Sources with a context tree

Definition 5.1. *A complete prefix dictionnary $\mathcal{D}$ on $\mathcal{X}$ (of cardinal K) is a cut of $\mathcal{X}^*$ (seen as a tree), that is, a finite subpart of $\mathcal{X}^*$ such that for all $x_{-\infty:-1}$, there is a unique m such that $x_{-m:-1} \in \mathcal{D}$. Let's call f its context function, defined by $f(x_{-\infty:-1}) = x_{-m:-1}$. Its depth, noted $l(\mathcal{D})$, is the maximum length of its elements (or, more simply, the depth of the cut).*

Suppose that we have reasons to think that $\mathbb{P}$ has a context tree $\mathcal{D}$ (or at least, can be succesfully approximated by such a law), that is, $\mathbb{P}$ is stationary and for all $x_{-\infty:n}$, $\mathbb{P}(X_n = x_n | X_{-\infty:n-1} = x_{-\infty:n-1}) = \mathbb{P}(X_n = x_n | f(X_{-\infty:n-1}) = f(x_{-\infty:n-1}))$.

We then want to estimate the conditional probabilities for all contexts in $\mathcal{D}$. Suppose that, for a context s, θ^s is the law on $\mathcal{X}$, conditionaly on the context s. Then, for a source with context $\mathcal{D}$, of parameter $(\theta^s)_{s\in\mathcal{D}}$,

$$\begin{aligned}\mathbb{P}_{\mathcal{D},\theta}(X_{1:n} = x_{1:n} | X_{-\infty:0} = x_{-\infty:0}) &= \prod_{i=1}^{n} \mathbb{P}_{\mathcal{D},\theta}(X_{1:n} = x_{1:n} | f(X_{-\infty:0}) = f(x_{-\infty:0})) \\ &= \prod_{s\in\mathcal{D}} \mathbb{P}_{\theta^s}(S_s(x_{1:n}; x_{-\infty:0}))\end{aligned}$$

where $\mathbb{P}_{\theta^s}$ is the law of a string of i.i.d. variables of law θ^s, and $S_s(x_{1:n}; x_{-\infty:0})$ is the substring of symbols in $x_{1:n}$ with context s.

The idea is to mix all those probabilities for all θ^s : for a prior distribution $\nu_{\mathcal{D}}(d\theta) = \prod_{s\in\mathcal{D}} \nu(d\theta^s)$, where ν is a measure on the set of possible θ^s, the simplex $\Theta = \{(\theta_1, \ldots, \theta_K) \in [0,1]^K | \sum \theta_i = 1\}$, we get:

$$\begin{aligned}KT_{\mathcal{D},\nu}(x_{1:n} | x_{-\infty:0}) &= \int_{\Theta^{\mathcal{D}}} \mathbb{P}_{\mathcal{D},\theta}(x_{1:n} | x_{-\infty:0}) \\ &= \prod_{s\in\mathcal{D}} \int_{\Theta} \mathbb{P}_{\theta^s}(S_s(x_{1:n}; x_{-\infty:0}))\nu(d\theta^s)\end{aligned}$$

A good choice for ν is a Dirichlet $\mathbb{D}(1/2, \ldots, 1/2)$ distribution: the Dirichlet Law with parameter $\alpha = (\alpha_1, \ldots, \alpha_K)$ is the law on Θ with density

$$f(\theta_1, \ldots, \theta_K) = \frac{\Gamma(\alpha_1 + \ldots + \alpha_K)}{\Gamma(\alpha_1) \ldots \Gamma(\alpha_K)} \prod_{i=1}^{K} \theta_i^{\alpha_i}$$

with respect to the Lebesgue density.

This distribution has nice properties, for example, an oracle inequality for the code-length. This choice gives the *Krichevski-Trofimov* estimator $KT_{\mathcal{D}} = KT_{\mathcal{D},\mathbb{D}(1/2,\ldots,1/2)}$ for sources with a context tree.

Lemma 5.1. *Let*

- $c_s^y(x_{1:n}|x_{-\infty:0})$ *denote the number of* y *in context* s*,*
- $C_s(x_{1:n}|x_{-\infty:0}) = \sum_y c_s^y(x_{1:n}|x_{-\infty:0})$*, the number of occurence of context* s*.*

Then:

$$KT_{\mathcal{D}}(x_{1:n}|x_{-\infty:0}) = \prod_{s\in\mathcal{D}} \frac{\Gamma(K/2)}{\Gamma(1/2)^K} \frac{\prod_{y\in\mathcal{X}} \Gamma(c_s^y(x_{1:n}|x_{-\infty:0}) + 1/2)}{\Gamma(C_s(x_{1:n}|x_{-\infty:0}) + 1/2)}$$

These quantities may be recursively computed by the following lemma (for a binary alphabet):

Lemma 5.2. *Let* $P_e(a, b)$ *denote the K-T estimator (giving the probability of having a* 0*) for a particular context* s*,* a *(resp* b*) representing the number of 0 (resp 1) seen in context* s*. Then* $P_e(0, 0) = 1$*, and for* $a \geq 0, b \geq 0$*,*

$$P_e(a+1, b) = \frac{a + 1/2}{a + b + 1} P_e(a, b) \qquad \textit{and} \qquad P_e(a, b+1) = \frac{b + 1/2}{a + b + 1} P_e(a, b).$$

The proof will not be presented here, but is quite easy, and may be found in [WST95]. The case of an alphabet of size K is identical, but much longer to write...

5.1.2 The Context Tree Weighting Method

When the context tree of our source is unknown, one solution is to mix over all possible context trees (of a certain maximum depth). The Context Tree Weighting methods consists in this idea: if π is a probability on context trees, we take

$$CTW(x_{1:n}) = \sum_{\mathcal{D}} \pi(\mathcal{D}) KT_{\mathcal{D}}(x_{1:n})$$

Typically, we will take for π a branching law, in which all nodes of depth less than a certain M has probability $\alpha \leq 1/K$ of having K daughters.

An important result is that this method is *universal*:

Theorem 5.1. *If* $(X_n)_{n\in\mathbb{N}}$ *is ergodic, stationary of law* $\mathbb{P}$*, then,* $\mathbb{P}$*-a.s.,*

$$\lim_{n\to\infty} -\frac{1}{n} \log CTW_\alpha(X_{1:n}) = H(\mathbb{P})$$

where $H(\mathbb{P})$ *is the entropy of* $\mathbb{P}$*.*

The great advantage of this method is that it may be recursively computed.

We will now present quickly how this can be done.

Let us denote, for each context s and $x \in \mathcal{X}$, x_s as the number of x seen in context s, and $P_e((x_s)_{x\in\mathcal{X}}, y)$ the corresponding K-T estimator (that is, the probability of having $y \in \mathcal{X}$ in context s).

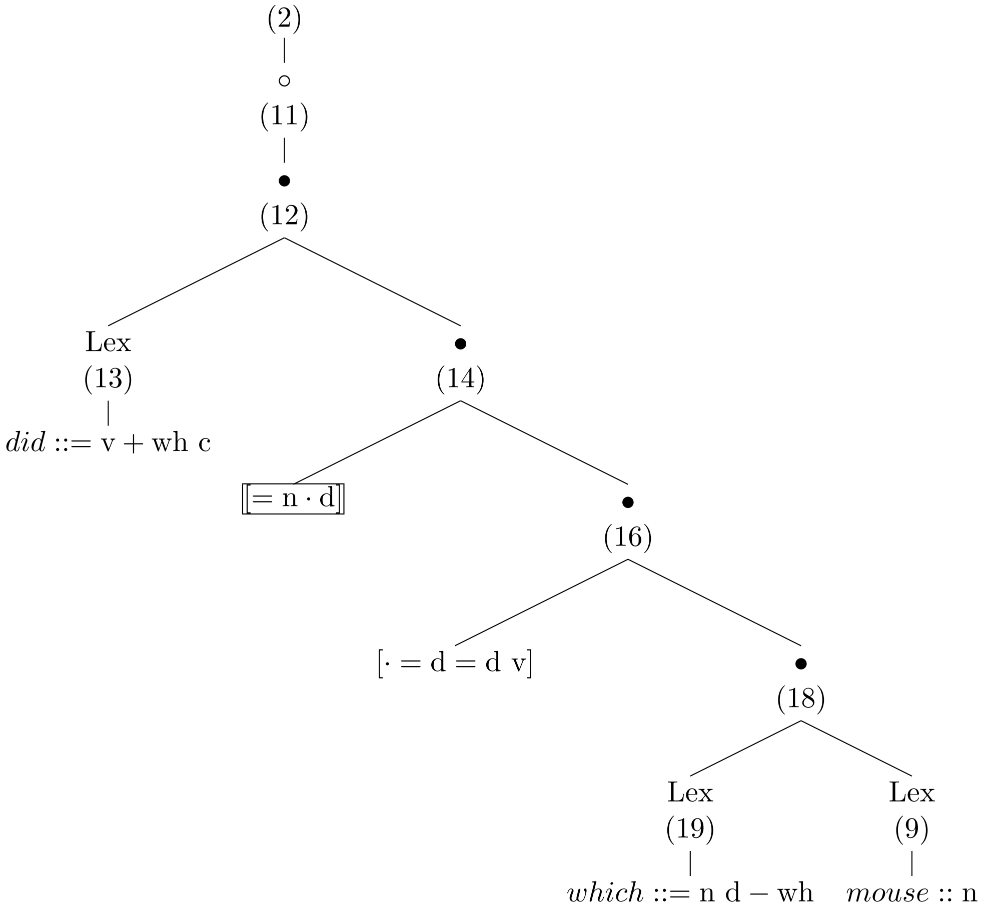
Definition 5.2. *To each node* s *of the context tree* $\mathcal{T}$ *of depth* M*, we assign a weighted probability* $\mathbb{P}_w^s$ *which is defined as*

$$\mathbb{P}_w^s(y) = \begin{cases} \frac{(K-1)P_e((x_s),y) + \prod_{\varphi\in\mathcal{X}} \mathbb{P}_w^{\varphi s}(y)}{K} & \textit{for } 0 \leq l(s) < M \\ P_e((x_s), y) & \textit{otherwise} \end{cases}$$

This construction has the expected property, that is, $\mathbb{P}_w^s(.) = CTW(.|s)$, for the $\alpha = 1/K$ mixture.

5.2 Application for the parser

This algorithm may be applied to our parser, to condition the rewriting rules. We can, with this method, condition on whatever we want, provided it appears as a linear string of symbols. An obvious (and perhaps a bit naïve) choice would be to condition on the string of rules already used in the parse. Another would be to condition on the string of rules *descending directly to the expanded node from the root*. For example, let's see, with our cats and mouses grammar (2) the sentence 'Which mouse did the cat eat' (3), when the parser will try to expand the node giving 'the cat' (at point 'Which mouse did · the cat eat'): it will be for example in state (index of the rules indicated in (.))



and will have already constructed the following string of rules: 2-11-12-14-16-18-19-9-13. It will condition on the string of rules 2-11-12-14, meaning that:

- it's the subject of a VP with moving object (14),
- it's a past sentence (12),
- it's an interrogative sentence (11 and 2)

Indeed, only the first seems relevant, but using all string of rules will condition first by the fact that it is a past sentence, that a mouse is involved, etc... and will have to go up 6 rules to know that it is a subject that is currently expanded... which seems the important information. This conditioning allows to condition roughly on the successive heads c-commanding the expanded node (plus movement information, which is difficult to get rid of). Of course we could want a different method for lexicalisation rules, where thematic and semantic information would be better... but such a method is difficult to implement, having to insure that conditioning still gives a proper distribution.

Three points seem important to precise:

- First, although the CTW algorithm works on any finite alphabet, it is much more efficient on binary ones. Ron Begleiter and Ran El-Yaniv discussed a method in [BEY06] to make the algorithm binary even in the case of an non-binary alphabet, by putting chains of CTWs (i.e. sorting the rules in a binary tree, and having a CTW algorithm for each branchement).
- Second, the distribution, as can be seen in the previous examples, is very sparse... a given context can be followed by two or three different rules, even one, while the alphabet is huge, with 23 rules in the cats and mouses grammar (which is very simple). Of course, more complete grammar will induce a lot more variability in the possible rules for a given node, but the number of rules will grow too. However, the restriction of rules expanding a given node can be implemented directly in the structure of the context tree of the CTW, provided we know in advance the grammar -which is of course the case.
- Finally, it is possible that, although the *grammar* allows for a choice of rewriting rules for a given category, there is in fact no such choice (or with vanishing probability). Then it is possible to use a slightly different base estimator instead of the Krichevski-Trofimov one: the zero-redundancy estimator $P_e^{ZR}(a,b)$ defined as:

$$P_e^{ZR}(a,b) = \begin{cases} \frac{1}{2}P_e(a,b) & \text{for } a > 0, b > 0 \\ \frac{1}{2}P_e(a,0) + \frac{1}{4} & \text{for } a > 0, b = 0 \\ \frac{1}{2}P_e(0,b) + \frac{1}{4} & \text{for } a = 0, b > 0 \\ 1 & \text{for } a = b = 0 \end{cases}$$

  This estimator better recognises sources generating only 0s or only 1s.

Conclusion

The method described here permits to see Minimalist Grammars as the more ‘classical’ and above all simpler Linear Context-Free Rewriting Systems (which don’t have movement, and generate sentences *top-down*), by taking a different point of view - considering derivation trees instead of derived trees. This enabled us to easily put a probability field on these grammars, and to parse them in a *top-down, incremental way*, giving a progressive parse as the words of the sentence are discovered. The probability field allowed us to implement a *beam-search* in the parser, pruning the different hypothesis to select only the more likely ones. This should accelerate the parser, while making it fail in identifying ‘garden-path’-type sentences. The use of more refined probabilistic tools as the CTW algorithm permits to have a better estimation of the real probability field, by conditionning the expanding rules by its context - here, the nature of the c-commanding heads, as required by the current linguistic theories.

References

[BEY06] Ron Begleiter and Ran El-Yaniv. Superior guarantees for sequential prediction and lossless compression via alphabet decomposition. *J. Mach. Learn. Res.*, 7:379–411, 2006.

[Cho95] Noam Chomsky. The minimalist program. *MIT Press, Cambridge, Massachusetts*, 1995.

[FP] Erik Franken and Marcel Peeters. Overview of the context tree weighting version 0.1 implementation. *"http://www.ele.tue.nl/ctw/download/ctw-v01_manual.pdf"*.

[Har01] Hendrik Harkema. Parsing minimalist grammars. 2001.

[Roa97] Brian Roark. Probabilistic top-down parsing and language modeling. *Logical aspects of computational linguistics*, 1997.

[Sta97] Edward Stabler. Derivational minimalism. *Logical aspects of computational linguistics*, 1997.

[WST95] Frans M.J. Willems, Yuri M. Shtarkov, and Tjalling J. Tjalkens. The context tree weighting method : basic properties. *IEEE-IT*, 41(3), 1995.