```
program
    var b: bool;
    var a: int;
    begin
        var b: int;
        b:=a;
    end;
    b:=a
end.
```

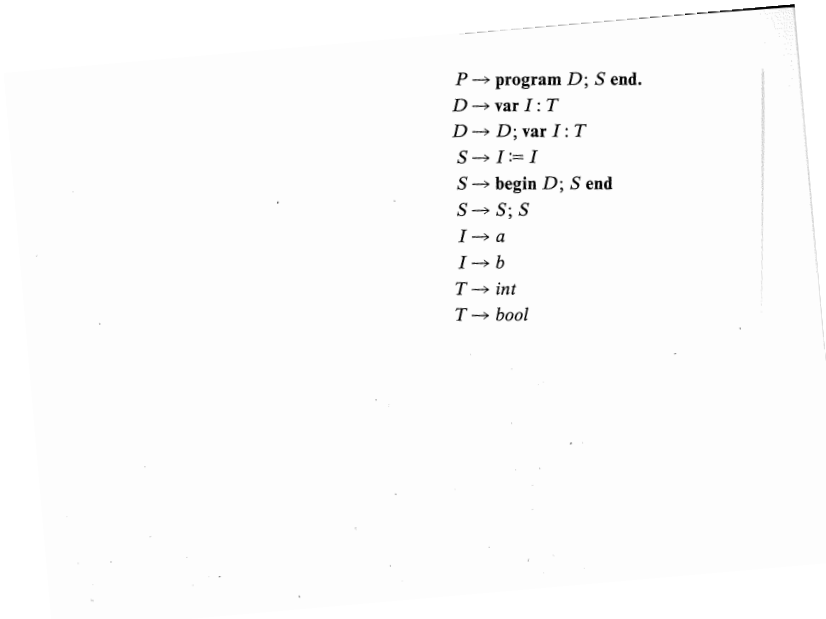


Figure: The context-free grammar generating a simple programming language.

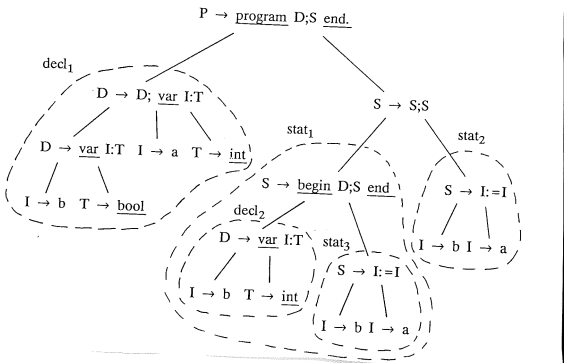


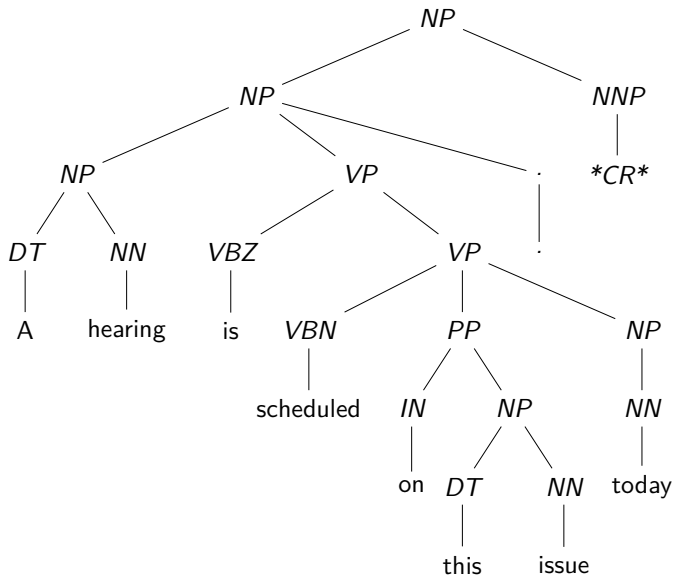
Figure: Syntactic description of the program.

Syntax of natural languages

> "Syntactic descriptions are concerned with three basic types of relationships in sentences: **sequence**, e.g. in English adjectives normally precede the nouns they modify, whereas in French they normally follow; **dependency**, i.e. relations between categories, e.g. prepositions may determine the morphological form (or case) of the nouns which depend on them in many languages, and verbs often determine the syntactic form of some of the other elements in a sentence —see below); and **constituency**, for example a noun phrase may consist of a determiner, an adjective and a noun." [Hutchins and Somers, 1992]

sequence: red wine - vin rouge
dependency: für den Vertrag, in dem Vertrag

Example of a phrase structure tree



Penn Treebank part-of-speech tags [Marcus et al., 1994]

No	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	to
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

Penn Treebank syntactic tagset [Marcus et al., 1994]

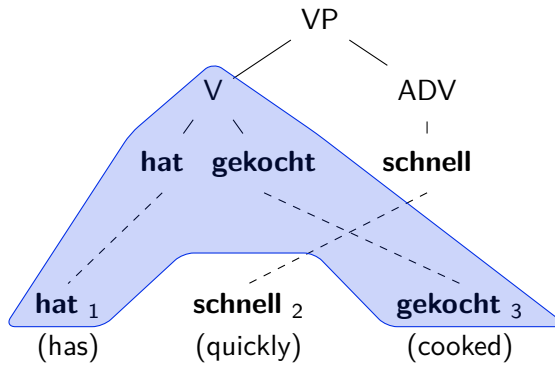
No	Tag	Description
1.	ADJP	Adjective phrase
2.	ADVP	Adverb phrase
3.	NP	Noun phrase
4.	PP	Prepositional phrase
5.	S	Simple declarative clause
6.	SBAR	Clause introduced by subordinating conjunction or 0
7.	SBARQ	Direct question introduced by wh-word or wh-phrase
8.	SINV	Declarative sentence with subject-aux inversion
9.	SQ	Subconstituent of SBARQ excluding wh-word or wh-phrase
10.	VP	Verb phrase
11.	WHADVP	wh-adverb phrase
12.	WHNP	wh-noun phrase
13.	WHPP	wh-propositional phrase
14.	X	Constituent of unknown or uncertain category

corpora: constituent treebanks

- Penn Treebank (for English, 2.499 stories from Wall Street Journal)
- TIGER Corpus (versions 2.1 and 2.2) (German, 50k sentences) [Brants et al., 2004]
- Japanese Verbmobil treebank (Japanese, 20k) (Kawata and Bartels, 2000),
- The Bosque part of the Floresta sinta(c)tica (Portuguese, 162.484 lexical units) (Afonso et al., 2002),
- Alpino treebank (Dutch, 150.000 words, newspaper articles) (van der Beek et al., 2002b; van der Beek et al., 2002a)
- ...

Linguistic Data Consortium `https://www.ldc.upenn.edu/`

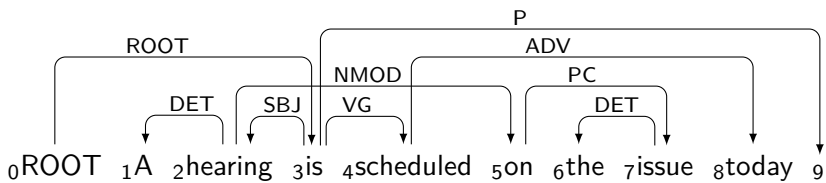
Discontinuous phrase structure tree



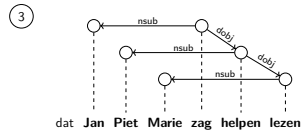
Dependency trees [de Marneffe and Manning, 2008]

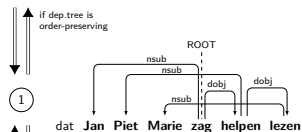
- dependency relation: advmod (adverb modifier)
  An adverb modifier of a word is a (non-clausal) adverb or adverb-headed phrase that serves to modify the meaning of the word.
  "Genetically modified food" advmod(modified, genetically)
  "less often" advmod(often, less)
- dependency relation: conj (conjunct)
  A conjunct is the relation between two elements connected by a coordinating conjunction, such as "and", "or", etc. We treat conjunctions asymmetrically: The head of the relation is the first conjunct and other conjunctions depend on it via the conj relation.
  "Bill is big and honest" conj(big, honest)
  "They either ski or snowboard" conj(ski, snowboard)

Dependency trees [Nivre, 2009]



Alternatives for the illustration of dependency trees. (i)

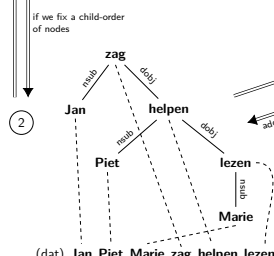


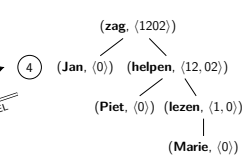


[Nivre, 2008]

[Kuhlmann, 2013]

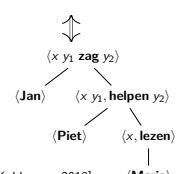
(This tree has a child order)





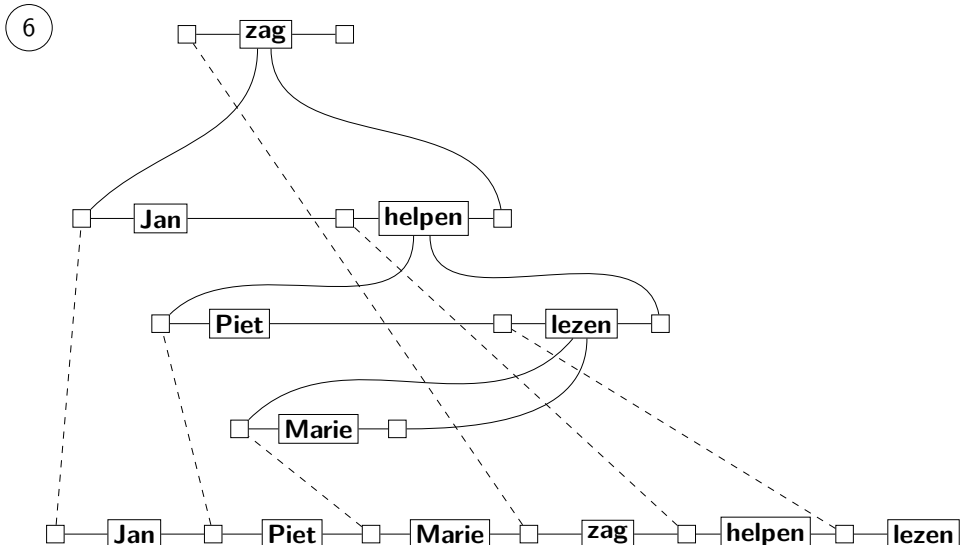
(order annotated tree)

[Kuhlmann and Möhl, 2007]



[Kuhlmann, 2013]

Alternatives for the illustration of dependency trees. (ii)



corpora: dependency treebanks

- TIGER (German, 50k) Forst et al. [2004],
- NEGRA (German, 20k sentences) [Skut et al., 1997],
- Prague Dependency Treebank (Czech, 88k (version 3.0)) [Böhmová et al., 2003],
- Prague Arabic Dependency Treebank (Arabic, 28k paragraphs (version 2.0)) (Hajič et al. 2004), (Smrž et al., 2002), (Smrž et al., 2008)
- Slovene Dependency Treebank (Slovene, 2k) (Džeroski et al. 2006)
- Danish Dependency Treebank (Danish, 5k) (Kromann, 2003),
- Talbanken (Swedish, 6k) (Teleman, 1974; Einarson, 1976; Nilsson, 2005),
- Metu-Sabanci Treebank (Turkish, 7k) Oflazer et al. 2003; Atalay et al. 2003)

Alternative illustrations of hybrid trees

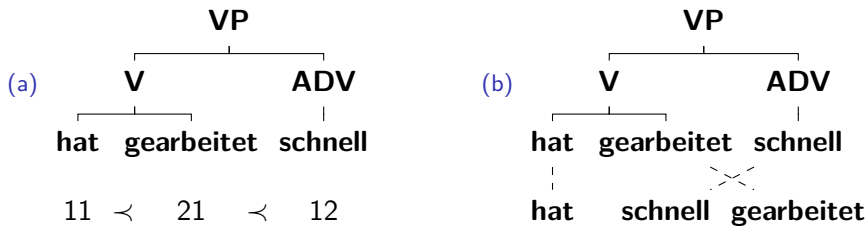
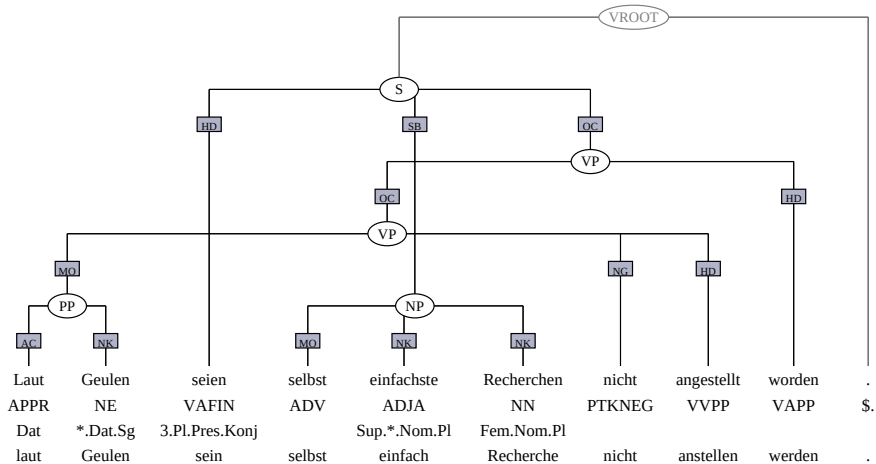


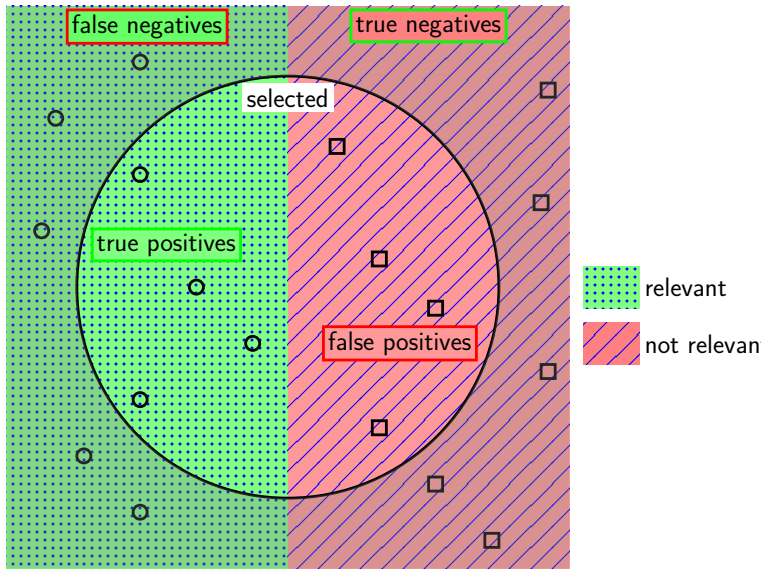
Figure: Phrase structure tree (a) with linear order on the set of leaves and (b) with a copy of the sentence and alignments.

A (discontinuous) phrase structure from the Tiger corpus [Brants et al., 2004]



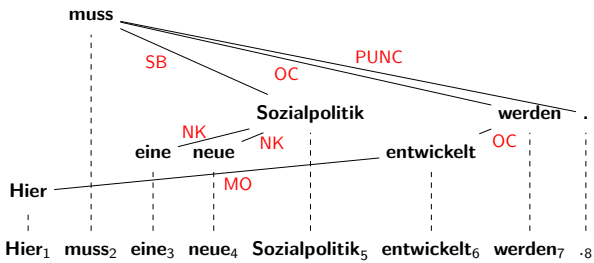
Each terminal is annotated by its POS-tag, morphological information, and its lemma. "The edges leading from one node to another are labeled according to the function of the child node in the constituent formed by the parent node." [Smith, 2003]

Binary classification

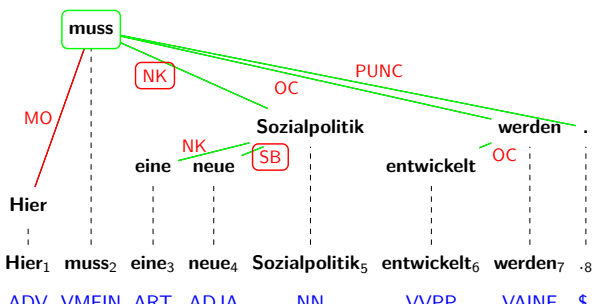


Selected	A	$\emptyset$	$\{Y_1, Y_2, Y_3, N_1, N_2, N_3\}$	$\{Y_1, Y_2, Y_3, N_1\}$	$\{Y_1, N_1, N_2, N_3\}$	$\{Y_1, Y_2, Y_3, Y_4, Y_5 N_1, N_2\}$
Precision	$\frac{5}{10} = 0.5$	undef.	$\frac{3}{6} = 0.5$	$\frac{3}{4} = 0.75$	$\frac{1}{4} = 0.25$	$\frac{5}{7} \approx 0.71$
Recall	$\frac{5}{5} = 1$	$\frac{0}{5} = 0$	$\frac{3}{5} = 0.6$	$\frac{3}{5} = 0.6$	$\frac{1}{5} = 0.2$	$\frac{5}{5} = 1$
Accuracy	$\frac{5}{10} = 0.5$	$\frac{5}{10} = 0.5$	$\frac{5}{10} = 0.5$	$\frac{7}{10} = 0.7$	$\frac{3}{10} = 0.3$	$\frac{8}{10} = 0.8$
F-measure	$\frac{2}{3} \approx 0.66$	undef.	$\frac{6}{11} \approx 0.54$	$\frac{2}{3} \approx 0.66$	$\frac{2}{9} \approx 0.22$	$\frac{5}{6} \approx 0.83$

Attachment scores



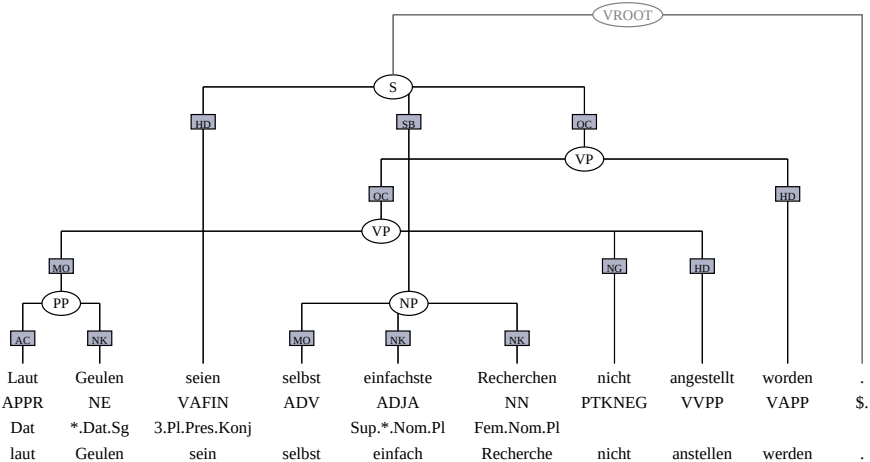
gold standard



UAS: 7 / 8

LAS: 5 / 8

Labeled precision, recall, F1



labeled brackets: $(\mathrm{VROOT}, \{1, \ldots, 10\}), (\mathrm{S}, \{1, \ldots, 9\}), (\mathrm{VP}, \{1, 2, 7, 8, 9\}), (\mathrm{VP}, \{1, 2, 7, 8\}), (\mathrm{NP}, \{4, 5, 6\})$

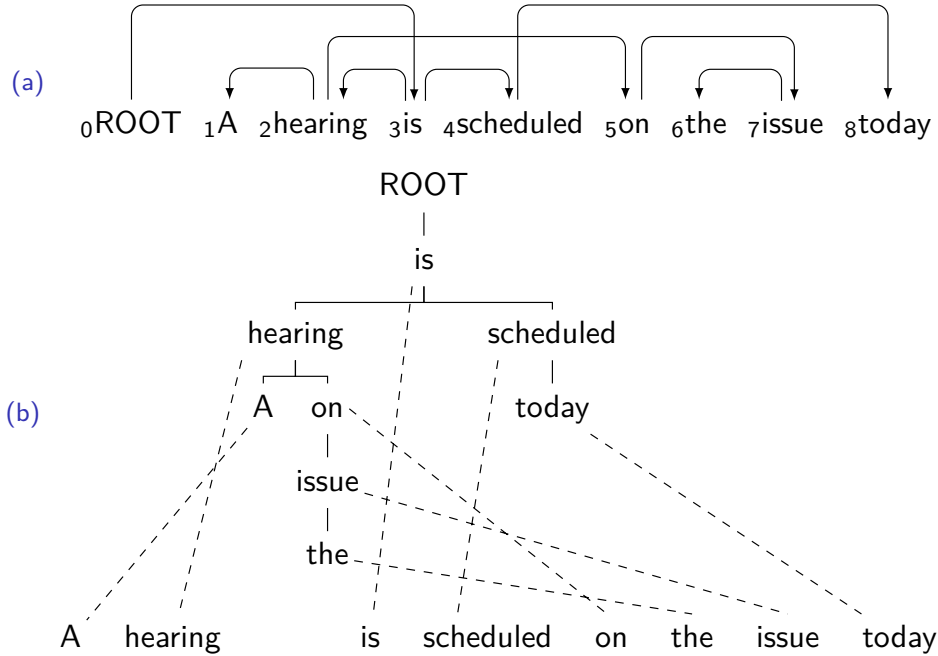


Figure: (a) Dependency graph. (b) Transformed dependency tree.

Algorithm 0.1 A simple deterministic dependency parser based on transition system $S = (C, T, c_s, C_t)$.

PARSE(o,**e**)

$c \leftarrow c_s(\mathbf{e})$
while $c \notin C_t$ **do**
  $t \leftarrow o(c)$; $c \leftarrow t(c)$
return dependency graph $h(c)$

How to find a "good" oracle? from [Nivre, 2009]

"... oracle can be approximated by a classifier trained on treebank data, a technique that has been used successfully in a number of systems (Yamada and Matsumoto, 2003; Nivre et al., 2004; Attardi, 2006)"

In [Kuhlmann et al., 2011] a dynamic approach to arc-eager for projective dependency trees is presented.

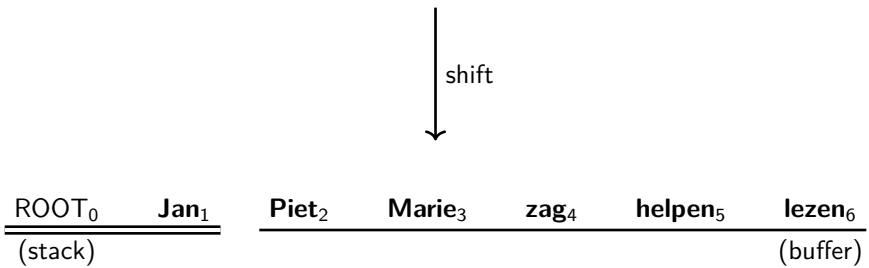
> *"... The basic idea, originally developed in the context of push-down automata (Lang, 1974; Tomita, 1986; Billot and Lang, 1989), is that while the number of computations of a transition-based parser may be exponential in the length of the input string, several portions of these computations, when appropriately represented, can be shared." [Kuhlmann et al., 2011]*

Computation of transition-based dependency parser

$\underline{\underline{\text{ROOT}_0}}$	**Jan**$_1$	**Piet**$_2$	**Marie**$_3$	**zag**$_4$	**helpen**$_5$	**lezen**$_6$
(stack)						(buffer)

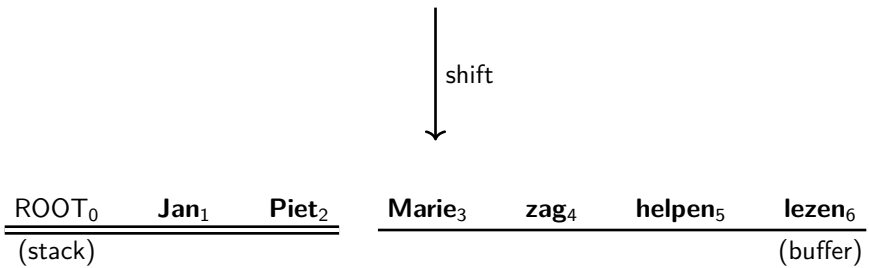
next action: shift

Computation of transition-based dependency parser



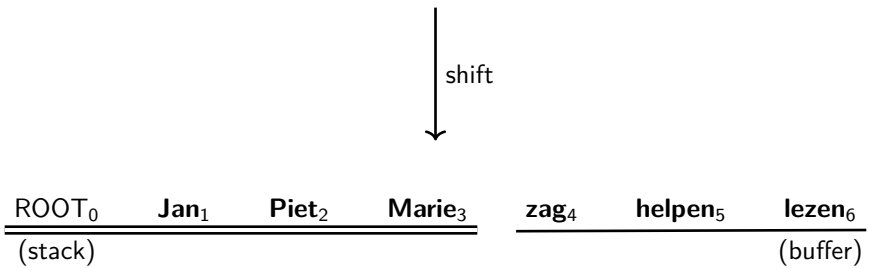
next action: shift

Computation of transition-based dependency parser

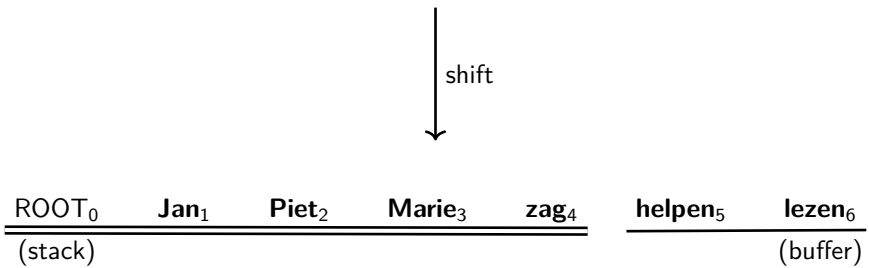


next action: shift

Computation of transition-based dependency parser

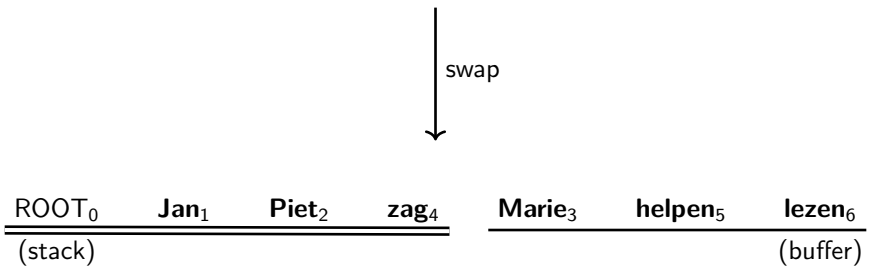


Computation of transition-based dependency parser



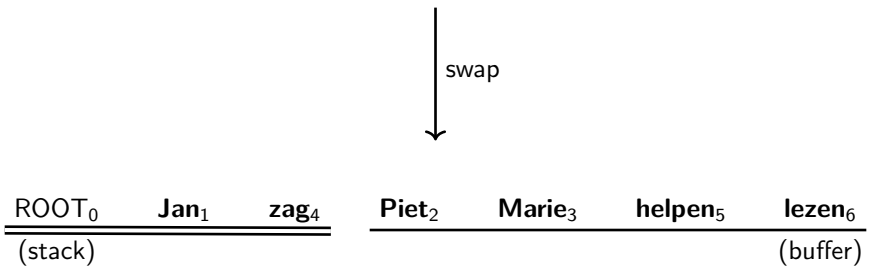
next action: swap

Computation of transition-based dependency parser



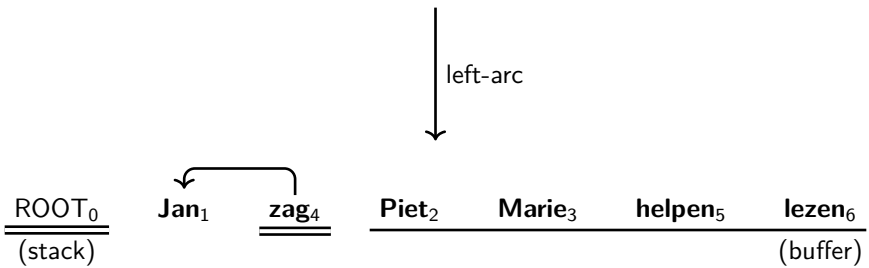
next action: swap

Computation of transition-based dependency parser



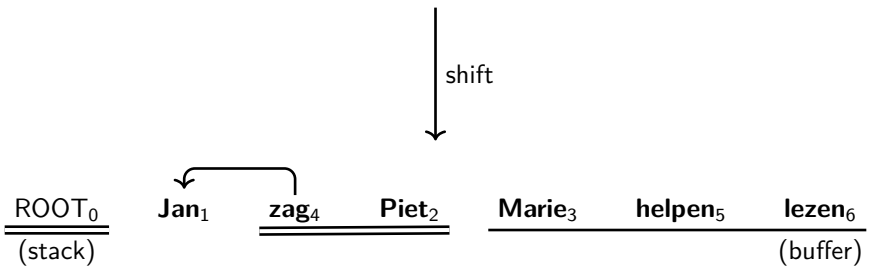
next action: left-arc

Computation of transition-based dependency parser



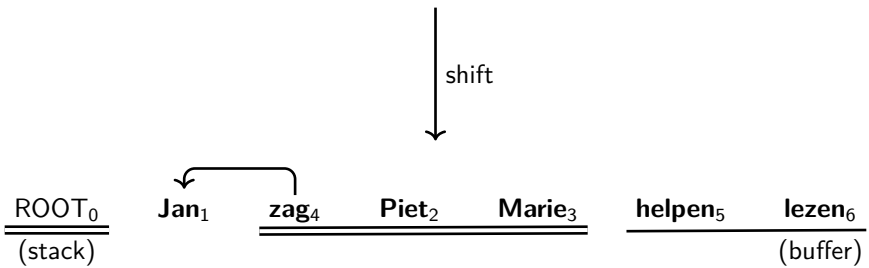
next action: shift

Computation of transition-based dependency parser



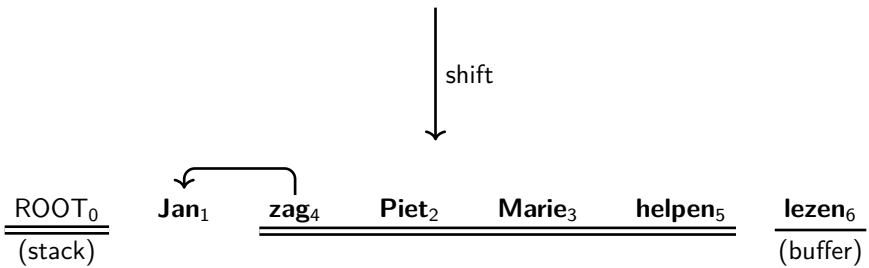
next action: shift

Computation of transition-based dependency parser



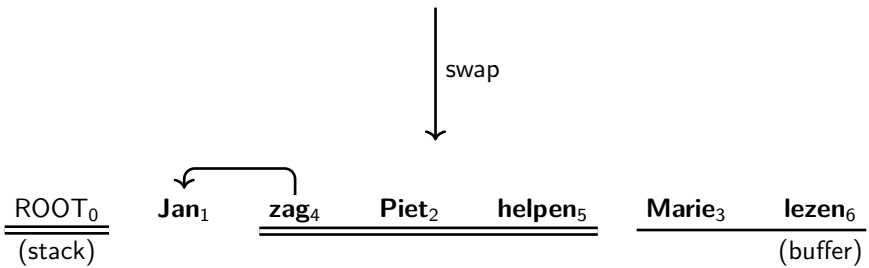
next action: shift

Computation of transition-based dependency parser



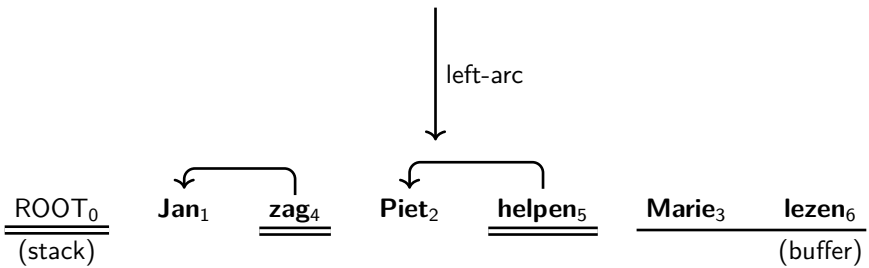
next action: swap

Computation of transition-based dependency parser



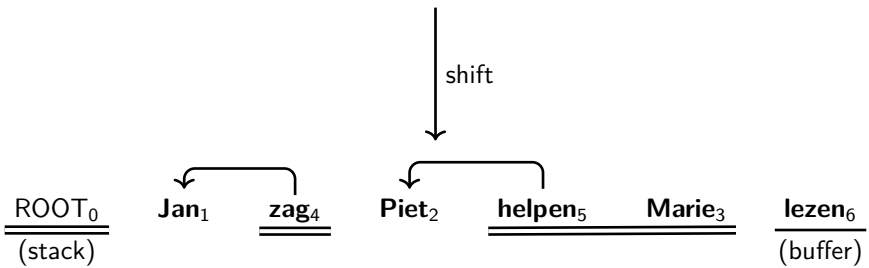
next action: left-arc

Computation of transition-based dependency parser



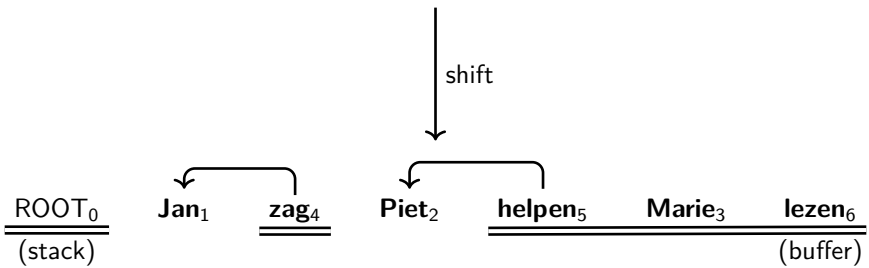
next action: shift

Computation of transition-based dependency parser



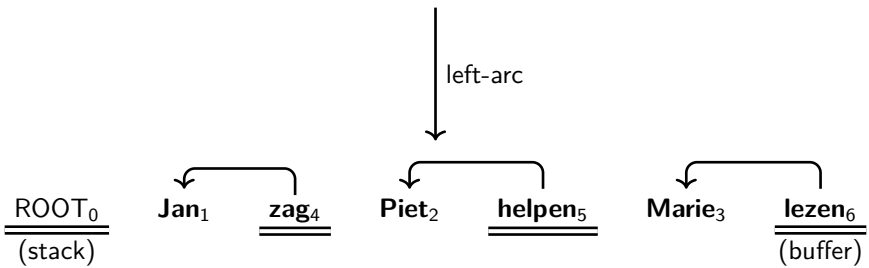
next action: shift

Computation of transition-based dependency parser



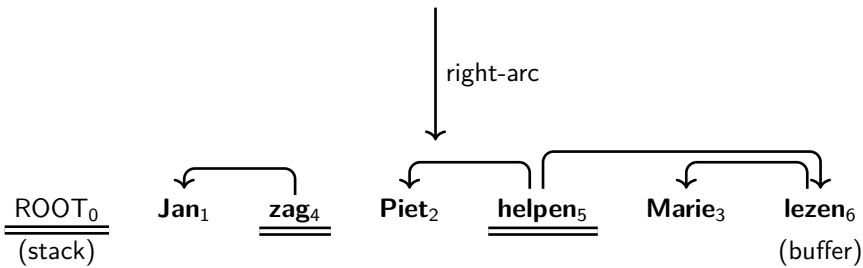
next action: left-arc

Computation of transition-based dependency parser



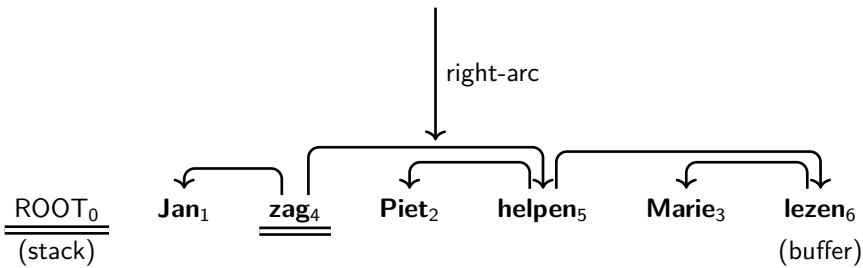
next action: right-arc

Computation of transition-based dependency parser



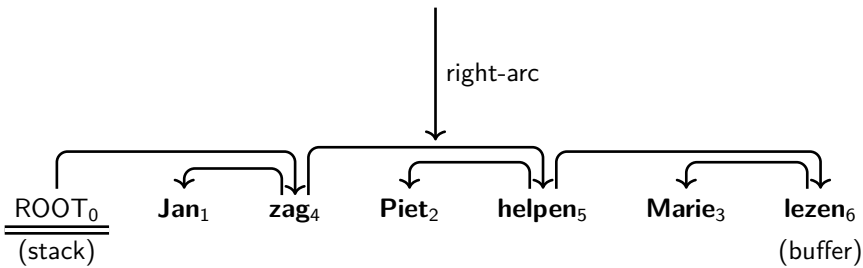
next action: right-arc

Computation of transition-based dependency parser



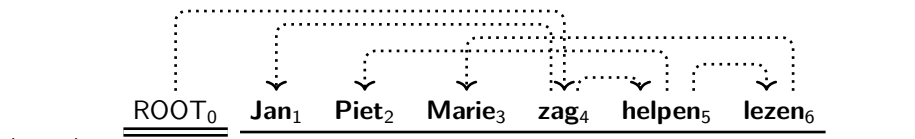
next action: right-arc

Computation of transition-based dependency parser



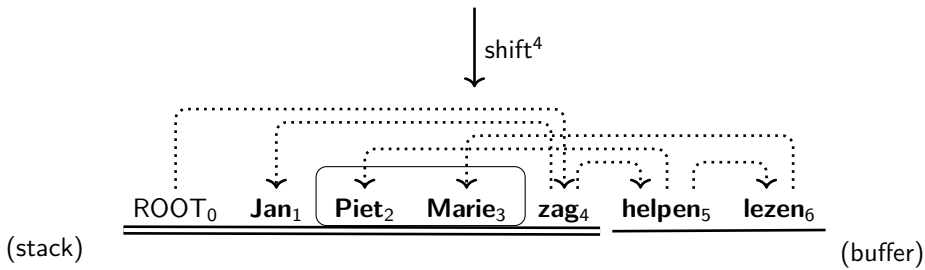
next action: done

Construction of transition sequence



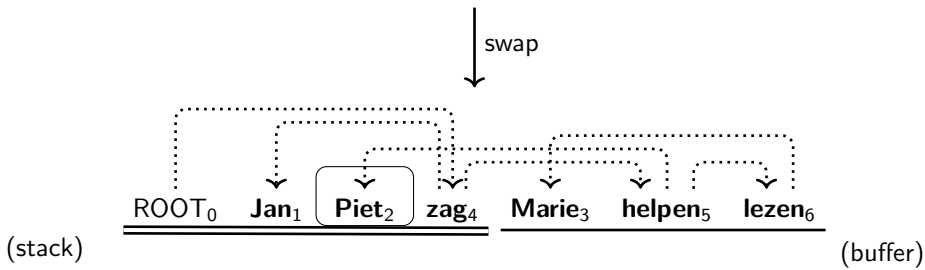
next action: shift[4]

Construction of transition sequence



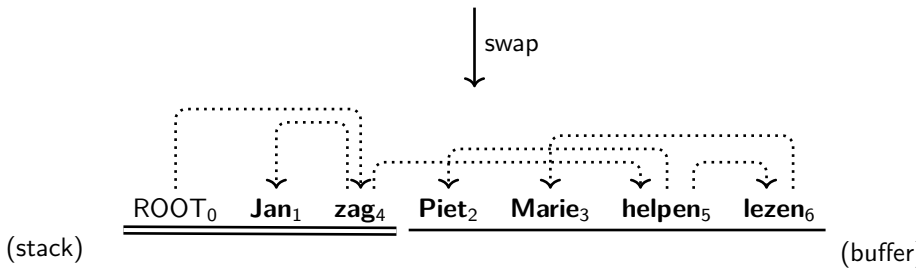
next action: swap

Construction of transition sequence



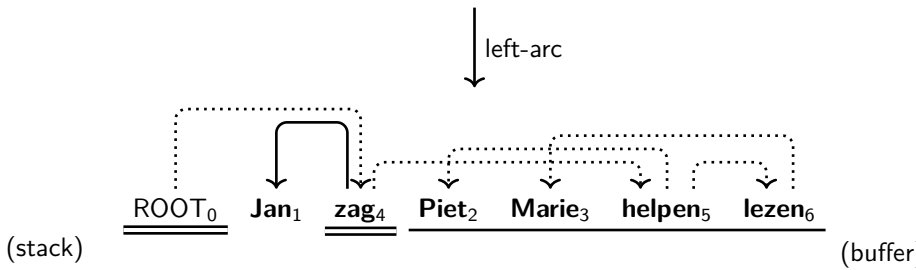
next action: swap

Construction of transition sequence



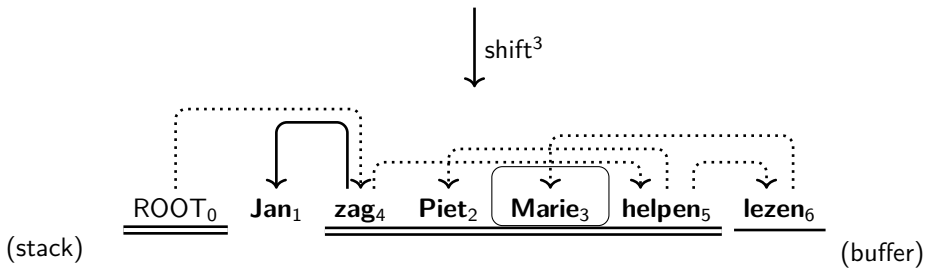
next action: left-arc

Construction of transition sequence



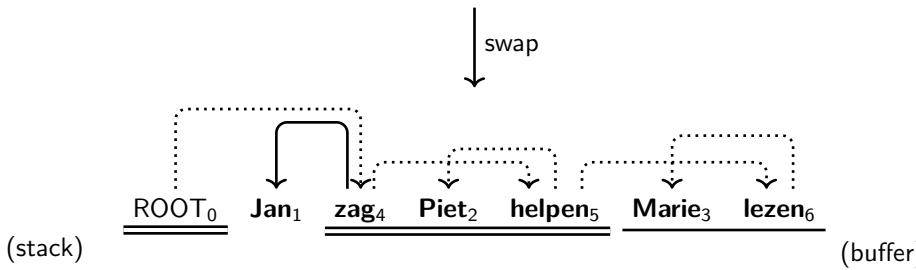
next action: shift[3]

Construction of transition sequence



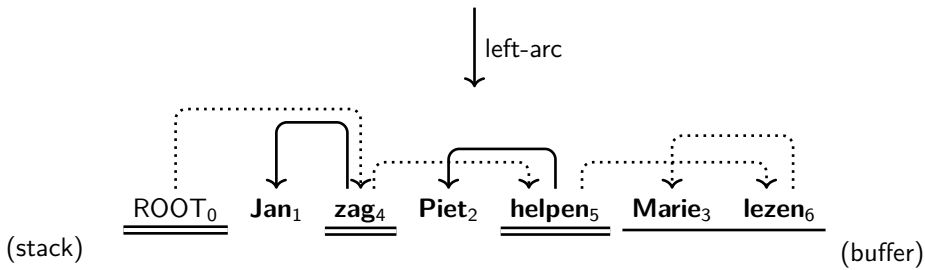
next action: swap

Construction of transition sequence



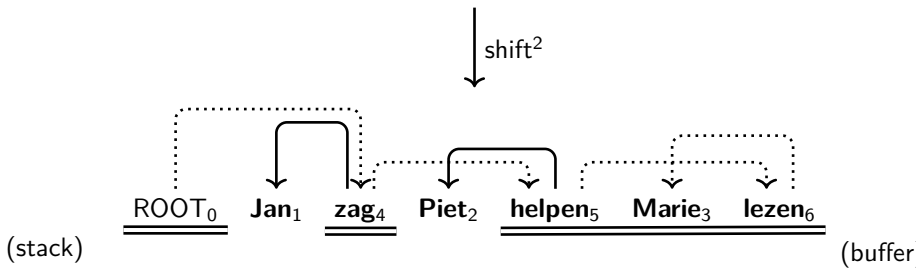
next action: left-arc

Construction of transition sequence



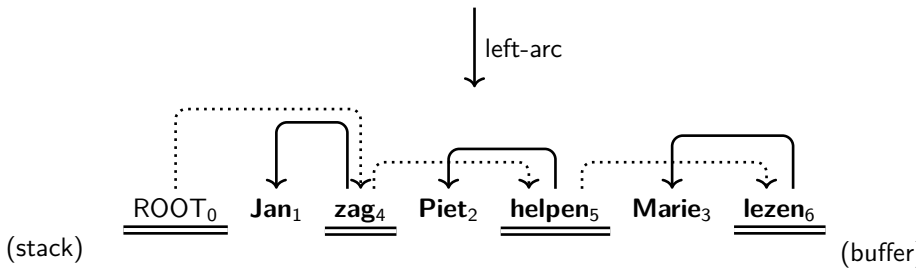
next action: shift[2]

Construction of transition sequence



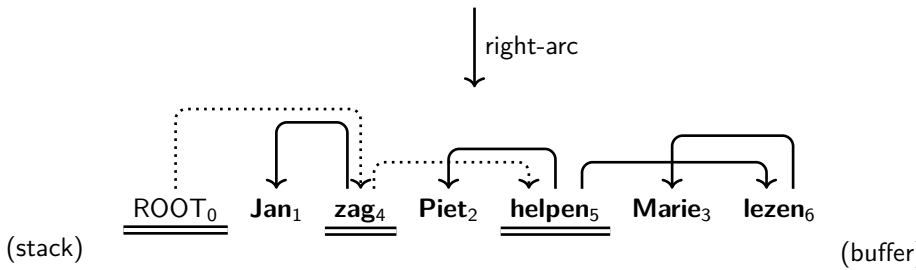
next action: left-arc

Construction of transition sequence



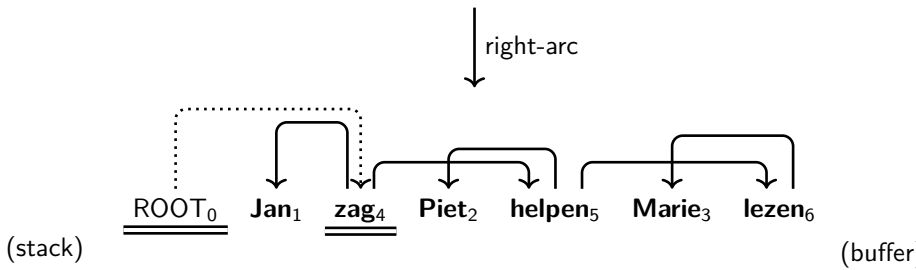
next action: right-arc

Construction of transition sequence



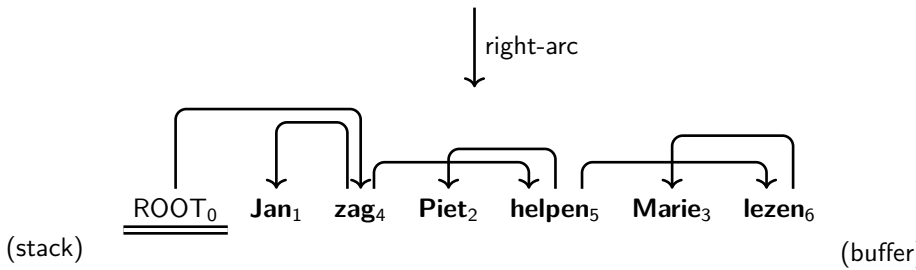
next action: right-arc

Construction of transition sequence



next action: right-arc

Construction of transition sequence



next action: done

Transition system for dependency parsing

A *transition system for dependency parsing* [Nivre, 2009] is a tuple $S = (C, T, c_s, C_t)$ where

- C is a set (*configurations*),
- T is a set of *transitions*; each transition is a partial function of type $t\colon C \to C$,
- c_s is an *initialization function*; c_s maps each sentence $\mathbf{e} = e_1 \dots e_n$ to a configuration $c \in C$, and
- $C_t \subseteq C$ is a set of *terminal configurations*.

Transition system for dependency parsing

A *transition system for dependency parsing* [Nivre, 2009] is a tuple $S = (C, T, c_s, C_t)$ where

- C is a set (*configurations*),
- T is a set of *transitions*; each transition is a partial function of type $t\colon C \to C$,
- c_s is an *initialization function*; c_s maps each sentence $\mathbf{e} = e_1 \dots e_n$ to a configuration $c \in C$, and
- $C_t \subseteq C$ is a set of *terminal configurations*.

Moreover, we require that each configuration c has the form (Σ, B, A) where

- $\Sigma \in \mathbb{N}^*$ (*stack*); the top of the stack is at the right,
- $B \in \mathbb{N}^*$ (*buffer*); we require that $\Sigma \cdot B$ is a finite sequence of pairwise distinct elements,
- A is a set of dependency arcs of the form (i, λ, j) where $i, j \in \mathbb{N}$ and $\lambda \in \Lambda$ is a dependency relation,
- each sentence $\mathbf{e} = e_1 \dots e_n$ is mapped by c_s to the configuration

$$c_s(\mathbf{e}) = ([0], [1, 2, \dots, n], \emptyset)$$

- each configuration of C_t has the form $([0], [], A)$ for some set A of dependency arcs.

Let $S = (C, T, c_s, C_t)$ be a transition system for dependency parsing and let $\mathbf{e} = e_1 \dots e_n$ be a sentence. A *transition sequence for* $\mathbf{e}$ is a sequence $C_{0,m} = (c_0, c_1, \dots, c_m)$ of configurations $c_i \in C$ such that

- $c_0 = c_s(\mathbf{e})$,
- for each $i \in [m]$ there is a $t \in T$ such that $c_i = t(c_{i-1})$, and
- $c_m \in C_t$.

Let $S = (C, T, c_s, C_t)$ be a transition system for dependency parsing and let $\mathbf{e} = e_1 \dots e_n$ be a sentence. A *transition sequence for* $\mathbf{e}$ is a sequence $C_{0,m} = (c_0, c_1, \dots, c_m)$ of configurations $c_i \in C$ such that

- $c_0 = c_s(\mathbf{e})$,
- for each $i \in [m]$ there is a $t \in T$ such that $c_i = t(c_{i-1})$, and
- $c_m \in C_t$.

Let $C_{0,m} = (c_0, c_1, \dots, c_m)$ be a transition sequence for $\mathbf{e}$ with $c_m = ([0], [], A)$. The *parse induced by* $C_{0,m}$, denoted by $h(c_m)$, is the dependency graph $(\{0\} \cup [n], A)$. We note that, in general, $h(c_m)$ need not correspond to a dependency tree, even not to a tree.

Let $\mathcal{G}$ be set of dependency graphs. A transition system S is *sound for* $\mathcal{G}$ if for every sentence $\mathbf{e}$ and transition sequence $C_{0,m}$ for $\mathbf{e}$, the parse $h(c_m)$ induced by $C_{0,m}$ is an element of $\mathcal{G}$. Moreover, S is *complete for* $\mathcal{G}$ if for every sentence $\mathbf{e}$ and dependency graph $h \in \mathcal{G}$ for $\mathbf{e}$, there is a transition sequence $C_{0,m}$ such that $h = h(c_m)$.

Let $\mathcal{G}$ be set of dependency graphs. A transition system S is *sound for* $\mathcal{G}$ if for every sentence $\mathbf{e}$ and transition sequence $C_{0,m}$ for $\mathbf{e}$, the parse $h(c_m)$ induced by $C_{0,m}$ is an element of $\mathcal{G}$. Moreover, S is *complete for* $\mathcal{G}$ if for every sentence $\mathbf{e}$ and dependency graph $h \in \mathcal{G}$ for $\mathbf{e}$, there is a transition sequence $C_{0,m}$ such that $h = h(c_m)$.

Let $S = (C, T, c_s, C_t)$ be a transition system for dependency parsing. An *oracle for* S is a mapping $o\colon C \to T$ such that c is in the domain of $o(c)$ for each $c \in C$.

Transitions				**Conditions**
LEFT-ARC$_l$	$([\sigma \mid i, j],\ B,\ A)$	$\Rightarrow$	$([\sigma \mid j],\ B,\ A \cup \{(j, l, i)\})$	$i \neq 0$
RIGHT-ARC$_l$	$([\sigma \mid i, j],\ B,\ A)$	$\Rightarrow$	$([\sigma \mid i],\ B,\ A \cup \{(i, l, j)\})$	
SHIFT	$([\sigma],\ [i \mid \beta],\ A)$	$\Rightarrow$	$([\sigma \mid i],\ [\beta],\ A)$	
SWAP	$([\sigma \mid i \mid j],\ [\beta],\ A)$	$\Rightarrow$	$([\sigma \mid j],\ [i \mid \beta],\ A)$	$0 < i < j$

Figure: Arc-standard transitions for non-projective dependency trees.

Theorem

[Nivre, 2009, Sect. 3.2] The transition system for dependency parsing which uses the arc-standard transitions shown in Figure 5 is sound and complete for the set of all dependency trees.

Productions of a regular tree grammar

$$
\begin{aligned}
\textbf{S} &\rightarrow \text{S}(\textbf{NP}, \textbf{VP}) \\
\textbf{NP} &\rightarrow \text{NP}(\textbf{Pronoun}) \mid \text{NP}(\textbf{Proper-Noun}) \mid \text{NP}(\textbf{Det}, \textbf{Nominal}) \\
\textbf{Nominal} &\rightarrow \text{Nominal}(\textbf{Noun}, \textbf{Nominal}) \mid \text{Nominal}(\textbf{Noun}) \\
\textbf{VP} &\rightarrow \text{VP}(\textbf{Verb}) \mid \text{VP}(\textbf{Verb}, \textbf{NP}) \mid \text{VP}(\textbf{Verb}, \textbf{NP}, \textbf{PP}) \mid \text{VP}(\textbf{Verb}, \textbf{PP}) \\
\textbf{PP} &\rightarrow \text{PP}(\textbf{Preposition}, \textbf{NP}) \\
\textbf{Noun} &\rightarrow \text{Noun(flight)} \mid \text{Noun(breeze)} \mid \text{Noun(trip)} \mid \text{Noun(morning)} \mid \ldots \\
\textbf{Verb} &\rightarrow \text{Verb(is)} \mid \text{Verb(prefer)} \mid \text{Verb(like)} \mid \text{Verb(need)} \mid \text{Verb(want)} \mid \text{Verb(fly)} \\
\textbf{Pronoun} &\rightarrow \text{Pronoun(me)} \mid \text{Pronoun(I)} \mid \text{Pronoun(you)} \mid \text{Pronoun(it)} \mid \ldots \\
\textbf{Proper-Noun} &\rightarrow \text{Proper-Noun(Alaska)} \mid \text{Proper-Noun(Baltimore)} \mid \ldots \\
\textbf{Det} &\rightarrow \text{Det(the)} \mid \text{Det(a)} \mid \text{Det(an)} \mid \text{Det(this)} \mid \text{Det(these)} \mid \text{Det(that)} \mid \ldots \\
\textbf{Preposition} &\rightarrow \text{Preposition(from)} \mid \text{Preposition(to)} \mid \text{Preposition(on)} \mid \ldots
\end{aligned}
$$

Derivation of a tree with a regular tree grammar

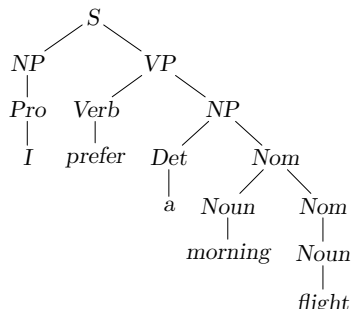
$$
\begin{aligned}
\mathbf{S} &\Rightarrow \mathrm{S}(\mathbf{NP}, \mathbf{VP}) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathbf{Pronoun}), \mathbf{VP}) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathrm{Pronoun}(\mathrm{I})), \mathbf{VP}) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathrm{Pronoun}(\mathrm{I})), \mathrm{VP}(\mathbf{Verb}, \mathbf{NP})) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathrm{Pronoun}(\mathrm{I})), \mathrm{VP}(\mathrm{Verb}(\mathrm{prefer}), \mathbf{NP})) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathrm{Pronoun}(\mathrm{I})), \mathrm{VP}(\mathrm{Verb}(\mathrm{prefer}), \mathrm{NP}(\mathbf{Det}, \mathbf{Nominal}))) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathrm{Pronoun}(\mathrm{I})), \mathrm{VP}(\mathrm{Verb}(\mathrm{prefer}), \mathrm{NP}(\mathrm{Det}(\mathrm{a}), \mathbf{Nominal}))) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathrm{Pronoun}(\mathrm{I})), \mathrm{VP}(\mathrm{Verb}(\mathrm{prefer}), \mathrm{NP}(\mathrm{Det}(\mathrm{a}), \mathrm{Nominal}(\mathbf{Noun}, \mathbf{Nominal})))) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathrm{Pronoun}(\mathrm{I})), \mathrm{VP}(\mathrm{Verb}(\mathrm{prefer}), \mathrm{NP}(\mathrm{Det}(\mathrm{a}), \mathrm{Nominal}(\mathrm{Noun}(\mathrm{morning}), \mathbf{Nominal})))) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathrm{Pronoun}(\mathrm{I})), \mathrm{VP}(\mathrm{Verb}(\mathrm{prefer}), \mathrm{NP}(\mathrm{Det}(\mathrm{a}), \mathrm{Nominal}(\mathrm{Noun}(\mathrm{morning}), \mathrm{Nominal}(\mathbf{Noun}))))) \\
&\Rightarrow \mathrm{S}(\mathrm{NP}(\mathrm{Pronoun}(\mathrm{I})), \mathrm{VP}(\mathrm{Verb}(\mathrm{prefer}), \mathrm{NP}(\mathrm{Det}(\mathrm{a}), \mathrm{Nominal}(\mathrm{Noun}(\mathrm{morning}), \\
&\qquad\qquad \mathrm{Nominal}(\mathrm{Noun}(\mathrm{flight})))))
\end{aligned}
$$

Context-free grammar G_{ATIS}

$$
\begin{aligned}
\text{S} &\rightarrow \text{NP VP} \\
\text{NP} &\rightarrow \text{Pronoun} \mid \text{Proper} - \text{Noun} \mid \text{Det Nominal} \\
\text{Nominal} &\rightarrow \text{Noun Nominal} \mid \text{Noun} \\
\text{VP} &\rightarrow \text{Verb} \mid \text{Verb NP} \mid \text{Verb NP PP} \mid \text{Verb PP} \\
\text{PP} &\rightarrow \text{Preposition NP} \\
\text{Noun} &\rightarrow \textbf{flight} \mid \textbf{breeze} \mid \textbf{trip} \mid \textbf{morning} \mid \ldots \\
\text{Verb} &\rightarrow \textbf{is} \mid \textbf{prefer} \mid \textbf{like} \mid \textbf{need} \mid \textbf{want} \mid \textbf{fly} \\
\text{Pronoun} &\rightarrow \textbf{me} \mid \textbf{I} \mid \textbf{you} \mid \textbf{it} \mid \ldots \\
\text{Proper} - \text{Noun} &\rightarrow \textbf{Alaska} \mid \text{Baltimore} \mid \textbf{Los Angeles} \mid \textbf{Chicago} \mid \ldots \\
\text{Det} &\rightarrow \textbf{the} \mid \textbf{a} \mid \textbf{an} \mid \textbf{this} \mid \textbf{these} \mid \textbf{that} \mid \ldots \\
\text{Preposition} &\rightarrow \textbf{from} \mid \textbf{to} \mid \textbf{on} \mid \textbf{near} \mid \ldots
\end{aligned}
$$

[Jurafsky and Martin, 2000, Figs. 9.2 and 9.3, p. 330]

Derivation tree of the context-free grammar G_{ATIS}



Rules of a lexicalized LCFRS

$$
\begin{aligned}
\text{root} &\to \langle x y_1 \ \textbf{zag} \ y_2 \rangle (\text{nsub}, \text{dobj}) \\
\text{dobj} &\to \langle x y_1, \textbf{helpen} \ y_2 \rangle (\text{nsub}, \text{dobj}) \\
\text{dobj} &\to \langle x, \textbf{lezen} \rangle (\text{nsub}) \\
\text{nsub} &\to \langle \textbf{Jan} \rangle \\
\text{nsub} &\to \langle \textbf{Piet} \rangle \\
\text{nsub} &\to \langle \textbf{Marie} \rangle \ .
\end{aligned}
$$

Rules and a derivation of a simple range concatenation grammar

$$\begin{aligned}
\mathrm{VP}(x_1 x_3 x_2) &\to \mathrm{V}(x_1, x_2)\ \mathrm{ADV}(x_3) \\
\mathrm{V}(\textbf{hat}, \textbf{gearbeitet}) &\to \varepsilon \\
\mathrm{ADV}(\textbf{schnell}) &\to \varepsilon
\end{aligned}$$

$$\begin{aligned}
&\mathrm{VP}(\textbf{hat schnell gearbeitet}) \\
\Rightarrow_G\ &\mathrm{V}(\textbf{hat}, \textbf{gearbeitet})\ \mathrm{ADV}(\textbf{schnell}) \\
\Rightarrow_G\ &\mathrm{ADV}(\textbf{schnell}) \\
\Rightarrow_G\ &\varepsilon
\end{aligned}$$

Example of an extented top-down tree transducer

The power of extended top-down tree transducers
(Jonathan Graehl, Mark Hopkins, Kevin Knight und Andreas Maletti)
SIAM J. Comput., 39(2):410–430, 2009

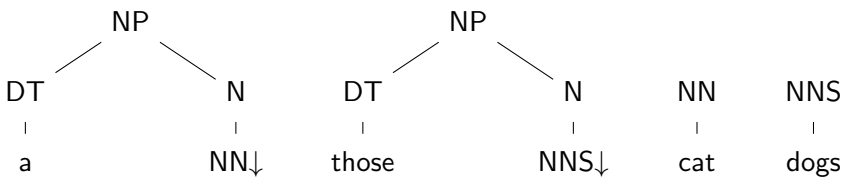
linear and nondeleting xtt $M = (Q, \Sigma, \Delta, I, R)$ with

- $Q = \{q, q_{\mathrm{S}}, q_{\mathrm{V}}, q_{\mathrm{NP}}\}$,
- $\Sigma = \{\mathrm{S}^{(2)}, \mathrm{NP}^{(2)}, \mathrm{VP}^{(2)}, \mathrm{DT}^{(1)}, \mathrm{N}^{(1)}, \mathrm{V}^{(1)}, \mathit{the}^{(0)}, \mathit{boy}^{(0)}, \mathit{saw}^{(0)}, \mathit{door}^{(0)}\}$,
- $\Delta = \{\mathrm{S}^{(2)}, \mathrm{S}'^{(2)}, \mathrm{CONJ}^{(1)}, \mathrm{V}^{(1)}, \mathrm{N}^{(1)}, \mathrm{NP}^{(1)}, \mathit{wa}\text{-}^{(0)}, \mathit{ra'aa}^{(0)}, \mathit{ateft}^{(0)}, \mathit{albab}^{(0)}\}$, and
- $I = \{q\}$.

$$
\begin{aligned}
q(x_1) &\to q_{\mathrm{S}}(x_1) && (r_1)\\
q(x_1) &\to \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), q_{\mathrm{S}}(x_1)) && (r_2)\\
q_{\mathrm{S}}(\mathrm{S}(x_1, \mathrm{VP}(x_2, x_3))) &\to \mathrm{S}'(q_{\mathrm{V}}(x_2), q_{\mathrm{NP}}(x_1), q_{\mathrm{NP}}(x_3)) && (r_3)\\
q_{\mathrm{V}}(\mathrm{V}(\mathit{saw})) &\to \mathrm{V}(\mathit{ra'aa}) && (r_4)\\
q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy}))) &\to \mathrm{NP}(\mathrm{N}(\mathit{ateft})) && (r_5)\\
q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door}))) &\to \mathrm{NP}(\mathrm{N}(\mathit{albab})) && (r_6)
\end{aligned}
$$

$$
\begin{aligned}
& q(\mathrm{S}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy})), \mathrm{VP}(\mathrm{V}(\mathit{saw}), \mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door})))))\\
\Rightarrow^{r_2}_M\ & \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), q_{\mathrm{S}}(\mathrm{S}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy})), \mathrm{VP}(\mathrm{V}(\mathit{saw}),\\
& \mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door}))))))\\
\Rightarrow^{r_3}_M\ & \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), \mathrm{S}'(q_{\mathrm{V}}(\mathrm{V}(\mathit{saw})), q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy}))),\\
& q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door})))))\\
\Rightarrow^{r_4}_M\ & \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), \mathrm{S}'(\mathrm{V}(\mathit{ra'aa}), q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{boy}))),\\
& q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door})))))\\
\Rightarrow^{r_5}_M\ & \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), \mathrm{S}'(\mathrm{V}(\mathit{ra'aa}), \mathrm{NP}(\mathrm{N}(\mathit{ateft})), q_{\mathrm{NP}}(\mathrm{NP}(\mathrm{DT}(\mathit{the}), \mathrm{N}(\mathit{door})))))\\
\Rightarrow^{r_6}_M\ & \mathrm{S}(\mathrm{CONJ}(\mathit{wa}\text{-}), \mathrm{S}'(\mathrm{V}(\mathit{ra'aa}), \mathrm{NP}(\mathrm{N}(\mathit{ateft})), \mathrm{NP}(\mathrm{N}(\mathit{albab}))))
\end{aligned}
$$

Tree substitution grammars: four elementary trees



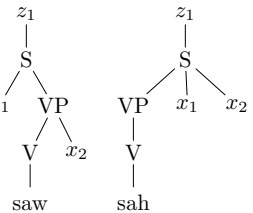
Synchronous context-free grammars

$$
\begin{array}{lrcl}
r_1: & q_{\mathrm{S}} & \rightarrow & \langle x_1 x_2 x_3 \ , \ x_1 x_2 x_3 \rangle (q_{\mathrm{SUB}}, q_{\mathrm{PRED}}, q_{\mathrm{OBJ}}) \\
r_2: & q_{\mathrm{S}} & \rightarrow & \langle x_1 x_2 x_3 \ , \ x_3 x_2 x_1 \rangle (q_{\mathrm{SUB}}, q_{\mathrm{PRED}}, q_{\mathrm{OBJ}}) \\
r_3: & q_{\mathrm{SUB}} & \rightarrow & \langle \text{he} \ , \ \text{er} \rangle \\
r_4: & q_{\mathrm{PRED}} & \rightarrow & \langle \text{saw} \ , \ \text{sah} \rangle \\
r_5: & q_{\mathrm{PRED}} & \rightarrow & \langle \text{drew} \ , \ \text{zeichnete} \rangle \\
r_6: & q_{\mathrm{PRED}} & \rightarrow & \langle \text{loved} \ , \ \text{liebte} \ , \ \varepsilon \rangle \\
r_7: & q_{\mathrm{OBJ}} & \rightarrow & \langle \text{the } x_1 \text{ house} \ , \ \text{das } x_1 \text{ Haus} \rangle (q_{\mathrm{ADJ}}) \\
r_8: & q_{\mathrm{OBJ}} & \rightarrow & \langle \text{the } x_1 \text{ triangle}, \ \text{das } x_1 \text{ Dreieck} \rangle (q_{\mathrm{ADJ}}) \\
r_9: & q_{\mathrm{ADJ}} & \rightarrow & \langle x_1 \ x_2 \ , \ x_1 \ x_2 \rangle (q_{\mathrm{ADV}}, q_{\mathrm{ADJ}}) \\
r_{10}: & q_{\mathrm{ADJ}} & \rightarrow & \langle \text{big} \ , \ \text{große} \rangle \\
r_{11}: & q_{\mathrm{ADV}} & \rightarrow & \langle \text{very} \ , \ \text{sehr} \rangle
\end{array}
$$

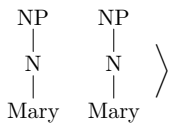
Synchronous tree substitution grammars (STSG)

$$
\begin{array}{lccl}
r_1: & o & \rightarrow & \langle \sigma(x_1, x_2), \sigma(x_2, x_1) \rangle (o, e) \\
r_2: & o & \rightarrow & \langle \sigma(x_2, x_1), \sigma(x_1, x_2) \rangle (o, e) \\
r_3: & o & \rightarrow & \langle \sigma(x_2, x_1), \sigma(x_2, x_1) \rangle (o, e) \\
r_4: & o & \rightarrow & \langle \alpha, \alpha \rangle \\
r_5: & e & \rightarrow & \langle \sigma(x_1, x_2), \sigma(x_2, x_1) \rangle (e, e) \\
r_6: & e & \rightarrow & \langle \sigma(x_1, x_2), \sigma(x_2, x_1) \rangle (o, o)
\end{array}
$$

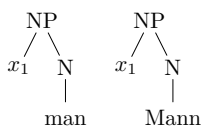
WSTAG [following Joshi and Schabes, 1997]

r_1: $q_1 \rightarrow \left\langle \right.$ 

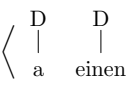
$\left. \right\rangle (q_2, q_2, f)$

r_2: $q_2 \rightarrow \left\langle \right.$ 

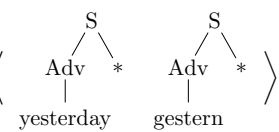
$\left. \right\rangle$

r_3: $q_2 \rightarrow \left\langle \right.$ 

$\left. \right\rangle (q_3)$

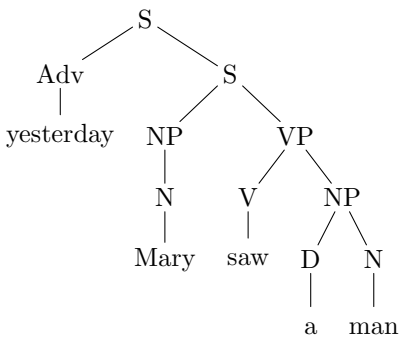
r_4: $q_3 \rightarrow \left\langle \right.$ 

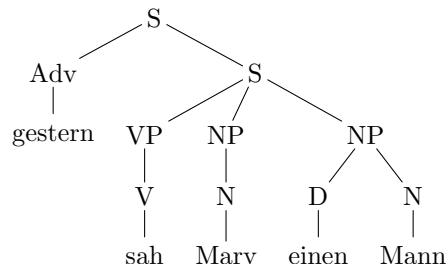
$\left. \right\rangle$

r_5: $f \rightarrow \left\langle \right.$ 

$\left. \right\rangle$

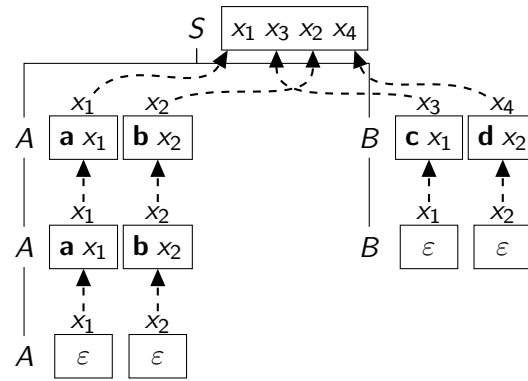
WSTAG: input tree and output tree





Derivation and derivation tree of a LCFRS

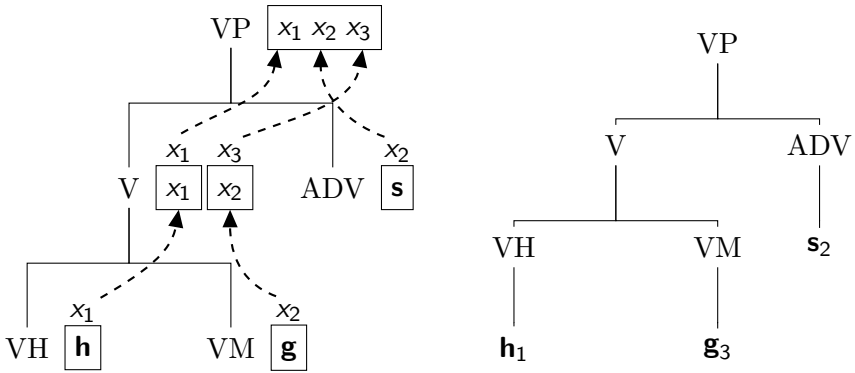
$$
\begin{array}{lll}
 & \mathsf{S}(\mathbf{a\ a\ c\ b\ b\ d}) & \\
\Rightarrow_G & A(\mathbf{a\ a}, \mathbf{b\ b}) & B(\mathbf{c}, \mathbf{d}) \\
\Rightarrow_G & A(\mathbf{a}, \mathbf{b}) & B(\mathbf{c}, \mathbf{d}) \\
\Rightarrow_G & A(\varepsilon, \varepsilon) & B(\mathbf{c}, \mathbf{d}) \\
\Rightarrow_G & & B(\mathbf{c}, \mathbf{d}) \\
\Rightarrow_G & & B(\varepsilon, \varepsilon) \\
\Rightarrow_G & & \varepsilon
\end{array}
$$



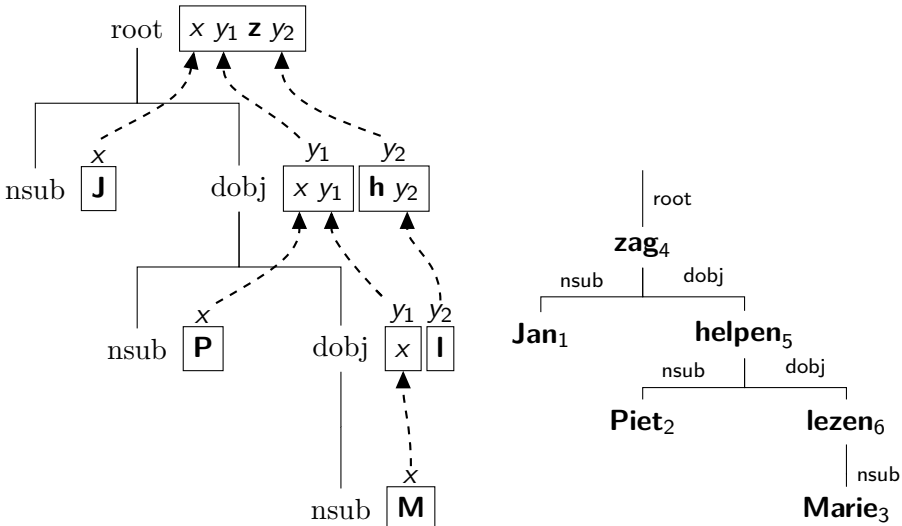
Traversal over derivation tree generated by simple or lexical. LCFRS

Input: derivation tree d of a simple LCFRS or a lexicalized LCFRS G
Output: hybrid tree h obtained by interpreting d
Variables: $q \in \mathrm{pos}(d)$ and $n \in \mathbb{N}$

1: $q \leftarrow \varepsilon$ (i.e., q points to the root of d), and $n = 1$
2: place $\bullet$ to the left end of the argument of S
3: **while** true **do**
4: let t be the symbol directly succeeding the bullet in the label at position q, i.e., $t \in \Delta \cup X \cup \{comma,)\}$
5: **if** $t \in \Delta$ **then**
6: move $\bullet$ one position to the right,
7: **if** G is simple **then** create a new leaf labeled with t below q and set $q' \leftarrow q1$
8: **if** G is a lexicalized LCFRS **then** set $q' \leftarrow q$
9: mark position q' as the n-th position in the linear order to be created, and set $n \leftarrow n + 1$
10: **else if** $t = x_j^{(i)}$ **then**
11: move $\bullet$ one position to the right, set $q \leftarrow qi$, and place a new instance of $\bullet$ in front of
12: the j-th argument of the nonterminal at q
13: **else if** $t \in \{comma,)\}$ **then**
14: remove $\bullet$ from the argument of position q and set q to its
15: predecessor if $q \neq \varepsilon$, otherwise break
16: **if** G is a lexicalized LCFRS **then** replace at each position q of d the label A by the unique terminal occurring in one of the arguments at q; moreover, label the edge from the predecessor of q to q by A (dependency relation)
17: remove all remaining parts of the dependency graph from d
18: **return** d



Derivation tree of the simple LCFRS G and its interpretation as (discontinuous) phrase structure tree.



Derivation tree of the lexicalized LCFRS G and its interpretation as (non-projective) dependency tree.

Grammar induction of a simple LCFRS grammar from a corpus of phrase structure trees with POS [Maier and Søgaard, 2008]

Input: a corpus c of phrase structure trees with POS

Output: a simple LCFRS in the form of a SRCG $G = (N, \Delta, Z, R)$ such that for each $h \in c$ there is a derivation tree d of G and h is the interpretation of d.

$N \leftarrow \{Z\} \cup$ set of syntactic categories $\cup$ POS

$\Delta \leftarrow$ vocabulary of **E**

$R \leftarrow \{Z(x_1) \rightarrow \xi(\varepsilon)(x_1) \mid (\xi, \mathrm{leaves}(\xi), \preceq) \in c\}$

for $h = (\xi, \mathrm{leaves}(\xi), \preceq)$ in c **do**

  for $p \in \mathrm{pos}(\xi) \setminus \mathrm{leaves}(\xi)$ **do**

    $k \leftarrow \mathrm{rk}_\Sigma(\xi(p))$ ▷ number of children of p in ξ

    if $p \in \mathrm{pos}_{\mathrm{POS}}(\xi)$ **then**

      $R \leftarrow R \cup \{\xi(p)(\xi(p1)) \rightarrow \varepsilon\}$

    else

      $J^{(0)} \leftarrow \mathrm{cover}(h, p), J^{(1)} \leftarrow \mathrm{cover}(h, p1), \ldots, J^{(k)} \leftarrow \mathrm{cover}(h, pk)$

      let $\mathrm{spans}(J^{(\ell)}) = (J_1^{(\ell)}, \ldots, J_{m_\ell}^{(\ell)})$ for each $\ell \in [k]_0$

      for $j \in [m_0]$ **do**

        $s_j \leftarrow \varepsilon$, $U \leftarrow J_j^{(0)}$

        while $U \neq \emptyset$ **do**

          let $\ell \in [k]$ and $i \in [m_\ell]$ such that $\min(U) = \min(J_i^{(\ell)})$

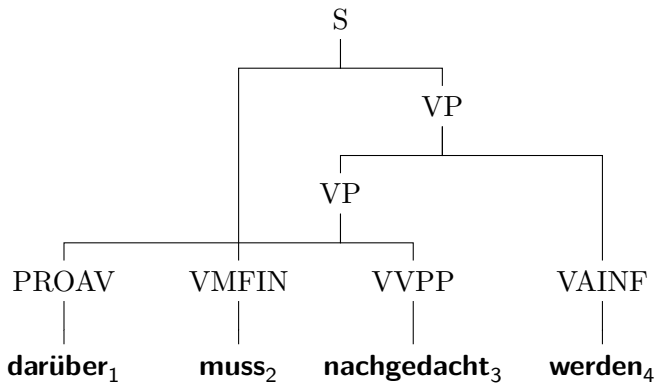
          $s_j \leftarrow s_j\ x_i^{(\ell)}$

          $U \leftarrow U \setminus J_i^{(\ell)}$

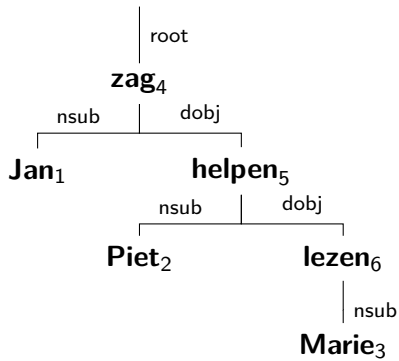
      $R \leftarrow R \cup \{\xi(p)(s_{1,m_0}) \rightarrow \xi(p1)(x_{1,m_1}^{(1)}) \ldots \xi(pk)(x_{1,m_k}^{(k)})\}$

return SRCG $G = (N, \Delta, Z, R)$

Discontinuous phrase structures tree



Dependency tree for the sentence "Jan Piet Marie zag helpen lezen"



Grammar induction of a lexicalized LCFRS grammar from a corpus of dependency trees with POS [Kuhlmann and Satta, 2009].

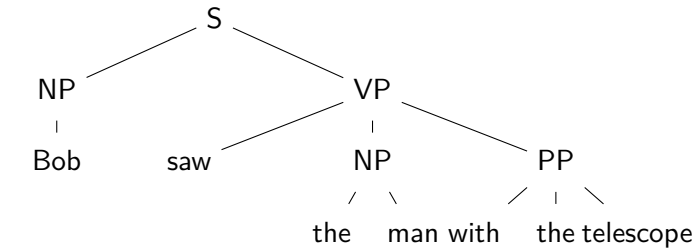
Input: a corpus c of dependency trees

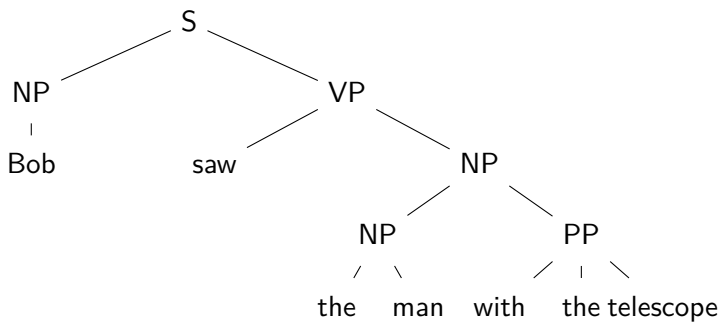
Output: a lexicalized LCFRS in the form of a SRCG $G = (N, \Delta, \mathrm{root}, R)$ such that for each $h \in c$ there is a derivation tree d of G and h is the interpretation of d

---

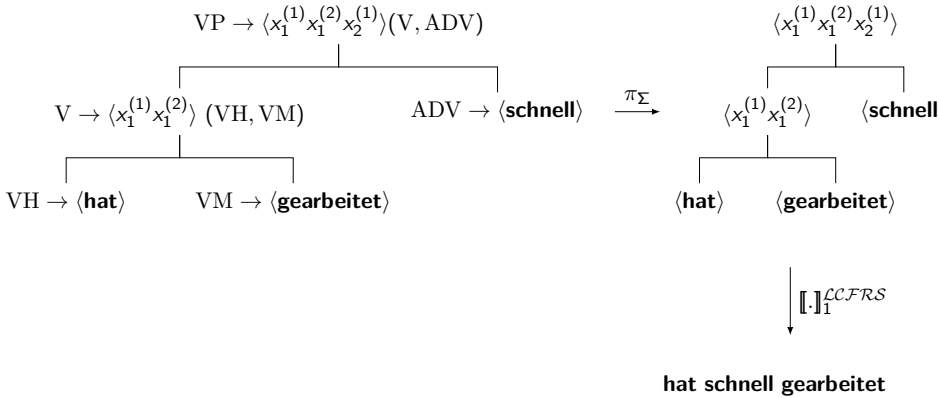
$N \leftarrow \mathrm{DEPREL} \cup \{\mathrm{root}\}$, $\Delta \leftarrow$ vocabulary of **E**, and $R \leftarrow \emptyset$
for $h = (\xi, \mathrm{lab}, \mathrm{pos}(\xi), \preceq)$ in c **do**
  for $p \in \mathrm{pos}(\xi)$ **do**
    $k \leftarrow \mathrm{rk}_{\Sigma}(\xi(p))$ ▷ number of children of p in ξ
    if $p \in \mathrm{leaves}(\xi)$ **then**
      $R \leftarrow R \cup \{\mathrm{deprel}(p)(\xi(p)) \rightarrow \varepsilon\}$
    else
      $J^{(0)} \leftarrow \mathrm{cover}(h, p), J^{(1)} \leftarrow \mathrm{cover}(h, p1), \ldots, J^{(k)} \leftarrow \mathrm{cover}(h, pk)$
      let $\mathrm{spans}(J^{(\ell)}) = (J_1^{(\ell)}, \ldots, J_{m_\ell}^{(\ell)})$ for each $\ell \in [k]_0$
      for $j \in [m_0]$ **do**
        $s_j \leftarrow \varepsilon$, $U \leftarrow J_j^{(0)}$
        while $U \neq \emptyset$ **do**
          if $\min(U) = \mathrm{ord}(p)$ **then**
            $s_j \leftarrow s_j\ \xi(p)$
            $U \leftarrow U \setminus \{\mathrm{ord}(p)\}$
          else
            let $\ell \in [k]$ and $i \in [m_\ell]$ such that $\min(U) = \min(J_i^{(\ell)})$
            $s_j \leftarrow s_j\ x_i^{(\ell)}$
            $U \leftarrow U \setminus J_i^{(\ell)}$
      $R \leftarrow R \cup \{\mathrm{deprel}(p)(s_{1,m_0}) \rightarrow \mathrm{deprel}(p1)(x_{1,m_1}^{(1)}) \ldots \mathrm{deprel}(pk)(x_{1,m_k}^{(k)})\}$
return SRCG $G = (N, \Delta, \mathrm{root}, R)$

Two different semantics of one sentence.



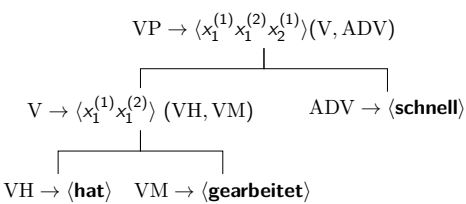


An abstract syntax tree and the images under π_Σ and $[\![.]\!]^{\mathcal{LCFRS}}$

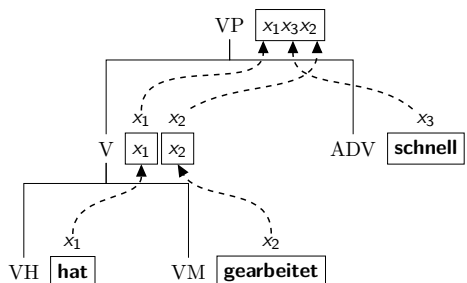


Example of an abstract syntax tree and the corresponding derivation tree

abstract syntax tree:



derivation tree:

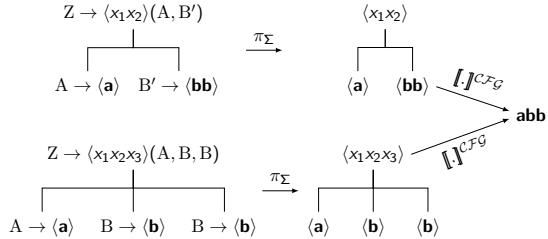


Generic inside-outside EM algorithm.

Prerequisites: unambiguous Y-grammar $G = (N, \Sigma, Z, R)$ and Y-algebra $\mathcal{Y} = (W, \varphi)$;
Input: finite W-corpus $c\colon W \to \mathbb{R}_{\geq 0}$
some initial probability distribution $p_0 : R \to \mathbb{R}_{\geq 0}$
Output: sequence $p_1, p_2, \ldots$ of probability distributions for R

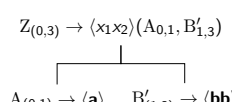
$i \leftarrow 0$
while true **do**
  // first, compute count c'
  let count $c' : R \to \mathbb{R}_{\geq 0}$ be defined by $c'(\rho) = 0$ for each $\rho \in R$
  for $w \in \mathrm{supp}(c)$ **do**
    $(Q, \Sigma, Z', R') \leftarrow (G \triangleright_\psi w)$
    compute the inside/outside weights in and out for the PY-grammar $((G \triangleright w), p_i \circ \psi)$
    if $\mathrm{in}(Z') \neq 0$ **then**
      for $\eta \in R', \eta = (q \to \sigma(q_1, \ldots, q_k))$ **do**
        $c'(\psi(\eta)) \leftarrow c'(\psi(\eta)) + c(w) \cdot \mathrm{in}(Z')^{-1} \cdot \mathrm{out}(q) \cdot p_i(\psi(\eta)) \cdot \mathrm{in}(q_1) \cdot \ldots \cdot \mathrm{in}(q_k)$
  // second, normalize count to a probability distribution
  for $A \in N$ **do**
    $s \leftarrow \sum_{\rho \in R_A} c'(\rho)$
    if $s = 0$ **then**
      for $\rho \in R_A$ **do**
        $p_{i+1}(\rho) \leftarrow p_i(\rho)$
    else
      for $\rho \in R_A$ **do**
        $p_{i+1}(\rho) \leftarrow s^{-1} \cdot c'(\rho)$
  output p_{i+1}
  $i \leftarrow i + 1$

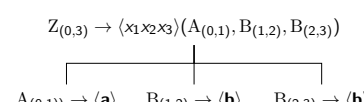
Example of reduct for a CFG G and a string w.



$$\mathrm{Z} \to \langle x_1 x_2 \rangle (\mathrm{AB}') \mid \langle x_1 x_2 x_3 \rangle (\mathrm{A}, \mathrm{B}, \mathrm{B}) \mid \langle x_1 \rangle (\mathrm{B})$$
$$\mathrm{A} \to \langle \mathbf{a} \rangle$$
$$\mathrm{B}' \to \langle \mathbf{bb} \rangle \qquad w = {}_0\mathbf{a}_1\mathbf{b}_2\mathbf{b}_3$$
$$\mathrm{B} \to \langle \mathbf{b} \rangle$$

$$\mathrm{Z}_{(0,3)} \to \langle x_1 x_2 \rangle (\mathrm{A}_{(0,1)} \mathrm{B}'_{1,3}) \qquad \mathrm{A}_{(0,1)} \to \langle \mathbf{a} \rangle$$
$$\mathrm{Z}_{(0,3)} \to \langle x_1 x_2 x_3 \rangle (\mathrm{A}_{(0,1)}, \mathrm{B}_{(1,2)}, \mathrm{B}_{(2,3)}) \qquad \mathrm{B}_{(1,2)} \to \langle \mathbf{b} \rangle$$
$$\mathrm{B}'_{(1,3)} \to \langle \mathbf{bb} \rangle \qquad \mathrm{B}_{(2,3)} \to \langle \mathbf{b} \rangle$$





Weighted deduction system for basic PLCFRS parsing.

SCAN: $\frac{}{p(\rho):[A,(i-1,i)]}$ if $\rho = (A \to \langle w_i \rangle)$ in R

RULE: $\frac{u_1 : [B_1,\vec{\kappa}_1],\ \ldots,\ u_k : [B_k,\vec{\kappa}_k]}{p(\rho)\cdot\prod_{i=1}^{k} u_i : [A,\sigma(\vec{\kappa}_1,\ldots,\vec{\kappa}_k)]}$ if $\rho = (A \to \sigma(B_1,\ldots,B_k))$ in R

Goal: $u : [Z,(0,n)]$

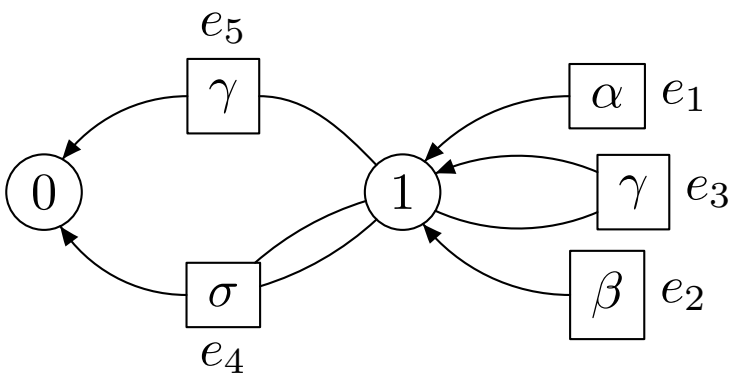
Weighted deductive parsing for LCFRS.

Input: PLCFRS (G, p) with monotone, ε-free, and simple $G = (N, \Delta, Z, R)$ and $w \in \Delta^*$

Output: either "$w \in L(G)$" or "$w \notin L(G)$"

1: add each item generated by SCAN to $\mathcal{A}$
2: **while** $\mathcal{A} \neq \emptyset$ **do**
3: $(u : [A, \vec{\kappa}]) \leftarrow \arg\max_{(u' : [A', \vec{\kappa'}]) \in \mathcal{A}} u'$
4: $\mathcal{A} \leftarrow \mathcal{A} \setminus \{(u : [A, \vec{\kappa}])\}$
5: $\mathcal{C} \leftarrow \mathcal{C} \cup \{(u : [A, \vec{\kappa}])\}$
6: **if** $(u : [A, \vec{\kappa}])$ is goal item **then** output "$w \in L(G)$" and stop
7: **else**
8: **for** each $v : [B, \vec{\eta}]$ deduced from $u : [A, \vec{\kappa}]$ and other items in $\mathcal{C}$ by RULE **do**
9: **if** there is no z with $(z : [B, \vec{\eta}])$ in $\mathcal{A} \cup \mathcal{C}$ **then**
10: $\mathcal{A} \leftarrow \mathcal{A} \cup \{(v : [B, \vec{\eta}])\}$
11: **else**
12: **if** $(z : [B, \vec{\eta}])$ in $\mathcal{A}$ for some z **then**
13: $\mathcal{A} \leftarrow (\mathcal{A} \setminus \{(z : [B, \vec{\eta}])\}) \cup \{(\max\{v, z\} : [B, \vec{\eta}])\}$
14: output "$w \notin L(G)$"

Example for a Hypergraph



Some useful functions specified in a functional programming style.

-- standard Haskell functions: list deconstructors, take operation

01 `head (x:xs) = x`

02 `tail (x:xs) = xs`

03 `take n xs = []` if `n == 0` or `xs == []`

04 `take n xs = (head xs):take (n-1) (tail xs)`

-- merge operation (lists in $\mathcal{L}$ should be disjoint)

05 `merge` $\mathcal{L}$ `= []` if $\mathcal{L} \setminus \{[]\} = \emptyset$

06 `merge` $\mathcal{L}$ `= m:merge` $(\{$`tail l` $\mid$ `l` $\in \mathcal{L}$, `l != []`, `head l == m`$\} \cup$
$\{$`l` $\mid$ `l` $\in \mathcal{L}$, `l != []`, `head l != m`$\})$

07 where `m` $= \min\{$`head l` $\mid$ `l` $\in \mathcal{L}$, `l != []`$\}$

-- top concatenation

08 $e($`l`$_1, \ldots,$`l`$_k)$ `= []` if `l`$_i$ `==  []` for some $i \in \{1, \ldots, k\}$

09 $e($`l`$_1, \ldots,$`l`$_k)$ `= e(head l`$_1, \ldots,$`head l`$_k)$`:merge` $\bigcup_{i \in \{1,\ldots,k\}} e($`l`$_1^i, \ldots,$`l`$_k^i)$

10 where $\mathtt{l}_j^i = \begin{cases} \mathtt{l}_j & \text{if } j < i \\ \mathtt{tail\ l}_j & \text{if } j = i \\ \mathtt{[head\ l}_j\mathtt{]} & \text{if } j > i \end{cases}$

Algorithm solving the 1-best-derivation problem.

Require Σ-hypergraph $H = (V, E)$, linear pre-order $\precsim$ fulfilling SP and CP.
Ensure `b v` $\in \min_1(H_v)$ for every $v \in V$ such that if `b v == [`$e(\xi_1, \ldots, \xi_k)$`]` for some $e = (v_1 \ldots v_k, \sigma, v) \in E$, then `b` v_i `== [`ξ_i`]` for every $i \in \{1, \ldots, k\}$.

01 `b = iter P0` $\{(\varepsilon, \alpha, v) \in E \mid \alpha \in \Sigma^{(0)}\}$
02 `P0 v = []`
03 `iter P` $\emptyset$ `= P`
04 `iter P c = iter P' c'`
05 where
06 ξ `= min c and` $\xi \in H_v$
07 `P' = (P//(v,[`ξ`]))`
08 `c' =` $\bigcup_{\substack{e=(v_1 \ldots v_k, \sigma, v') \in E \\ \texttt{P' v' == []}}} e(\texttt{P' } v_1, \ldots, \texttt{P' } v_k)$

Algorithm solving the n-best-derivations problem.

Require Σ-hypergraph $H = (V, E)$, linear pre-order $\precsim$ fulfilling SP and CP.
Ensure $(\texttt{take n (p v)}) \in \min_n(H_v)$ for every $v \in V$ and $n \in \mathbb{N}$.

01 `p v = []` if `b v == []`
02 `p v =` $e(\xi_1, \ldots, \xi_k)$
`:merge` $(\{\texttt{tail } e(\texttt{p v}_1, \ldots, \texttt{p v}_k)\} \cup$
$\{e'(\texttt{p v}'_1, \ldots, \texttt{p v}'_k) \mid e' = (v'_1 \ldots v'_k, \sigma', \texttt{v}) \in E,\ e' \neq e\})$
if `b v ==` $[e(\xi_1, \ldots, \xi_k)]$ `where` $e = (v_1 \ldots v_k, \sigma, \texttt{v})$

Binarizing a LCFRS.

Input: LCFRS $G = (N, \Delta, Z, R)$
Output: binary LCFRS $G' = (N', \Delta, Z, R')$ such that $L(G) = L(G')$

1: **for** each rule r of the form $A(w_1, \ldots, w_n) \to A_1(x_1, \ldots, x_{l_1}), \ldots, A_k(x_{m-l_k+1}, \ldots, x_m)$ with $k > 2$ **do**
2: remove r from R
3: $P \leftarrow \emptyset$
4: take new nonterminals $C_1, \ldots, C_{k-2}$
5: add the rule $A(w_1, \ldots, w_n) \to A_1(x_1, \ldots, x_{l_1}), C_1(\vec{\alpha_1})$ where $\vec{\alpha_1}$ is the reduction of $(w_1, \ldots, w_n)$ by $(x_1, \ldots, x_{l_1})$.
6: **for** each i with $1 \leq i \leq k-3$ **do**
7: add the rule $C_i(\vec{\alpha_i}) \to A_{i+1}(x_{m_i+1}, \ldots, x_{m_{i+1}})C_{i+1}(\vec{\alpha_{i+1}})$ to P where $\vec{\alpha_{i+1}}$ is the reduction of $\vec{\alpha_i}$ by $(x_{m_i+1}, \ldots, x_{m_i})$.
8: add the rule $C_{k-2}(\vec{\alpha_{k-2}}) \to A_{k-1}(x_{m_{k-2}+1}, \ldots, x_{m_{k-1}})A_k(x_{m_{k-1}+1}, \ldots, m_k)$ to P
9: **for** each rule $r \in P$ **do**
10: replace right-hand side arguments with length greater 1 by new variables (consistently on both sides of r) and add the result to R'
11: **return** LCFRS $G = (N', \Delta, Z, R')$

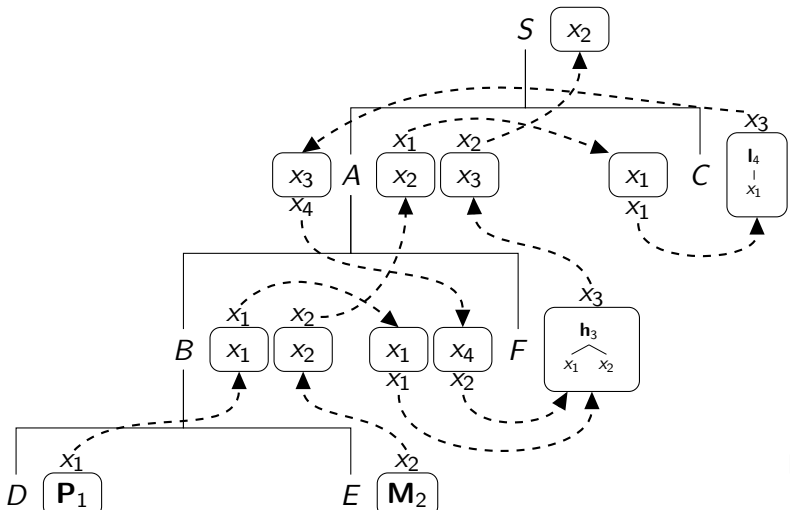
A simple definite clause program (SDCP)

$$
\begin{array}{lcl}
S(\varepsilon; x_2) & \rightarrow & A(x_3; x_1, x_2)\ C(x_1; x_3) \\
A(x_4; x_2, x_3) & \rightarrow & B(\varepsilon; x_1, x_2)\ F(x_1, x_4; x_3) \\
B(\varepsilon; x_1, x_2) & \rightarrow & D(\varepsilon; x_1)\ E(\varepsilon; x_2) \\
D(\varepsilon; \textbf{Piet}) & \rightarrow & \varepsilon \\
E(\varepsilon; \textbf{Marie}) & \rightarrow & \varepsilon \\
F(x_1, x_2; \textbf{helpen}(x_1, x_2)) & \rightarrow & \varepsilon \\
C(x_1; \textbf{lezen}(x_1)) & \rightarrow & \varepsilon
\end{array}
$$

A derivation of a SDCP

$$
\begin{array}{lll}
 & S(\varepsilon;\mathbf{h}(\mathbf{P},\mathbf{I}(\mathbf{M}))) & \\
\Rightarrow & A(\mathbf{I}(\mathbf{M});\mathbf{M},\mathbf{h}(\mathbf{P},\mathbf{I}(\mathbf{M})))\ C(\mathbf{M};\mathbf{I}(\mathbf{M})) & (x_1=\mathbf{M},\ x_2=\mathbf{h}(\mathbf{P},\mathbf{I}(\mathbf{M})),\ x_3=\mathbf{I}(\mathbf{M})) \\
\Rightarrow & B(\varepsilon;\mathbf{P},\mathbf{M})\ F(\mathbf{P},\mathbf{I}(\mathbf{M});\mathbf{h}(\mathbf{P},\mathbf{I}(\mathbf{M})))\ C(\mathbf{M};\mathbf{I}(\mathbf{M})) & (x_1=\mathbf{P},\ x_2=\mathbf{M},x_3=\mathbf{h}(\mathbf{P},\mathbf{I}(\mathbf{M})),\ \ x_4=\mathbf{I}(\mathbf{M}),\ x=\mathbf{P}) \\
\Rightarrow & D(\varepsilon;\mathbf{P})\ E(\varepsilon;\mathbf{M})\ F(\mathbf{P},\mathbf{I}(\mathbf{M});\mathbf{h}(\mathbf{P},\mathbf{I}(\mathbf{M})))\ C(\mathbf{M};\mathbf{I}(\mathbf{M})) & (x_1=\mathbf{P},\ x_2=\mathbf{M}) \\
\Rightarrow & E(\varepsilon;\mathbf{M})\ F(\mathbf{P},\mathbf{I}(\mathbf{M});\mathbf{h}(\mathbf{P},\mathbf{I}(\mathbf{M})))\ C(\mathbf{M};\mathbf{I}(\mathbf{M})) & \\
\Rightarrow & F(\mathbf{P},\mathbf{I}(\mathbf{M});\mathbf{h}(\mathbf{P},\mathbf{I}(\mathbf{M})))\ C(\mathbf{M};\mathbf{I}(\mathbf{M})) & \\
\Rightarrow & C(\mathbf{M};\mathbf{I}(\mathbf{M})) & (x_1=\mathbf{P},\ x_2=\mathbf{I}(\mathbf{M})) \\
\Rightarrow & \varepsilon &
\end{array}
$$

Derivation tree of a sDCP.



$\mathbf{P}_1\ \mathbf{M}_2\ \mathbf{h}_3\ \mathbf{l}_4$

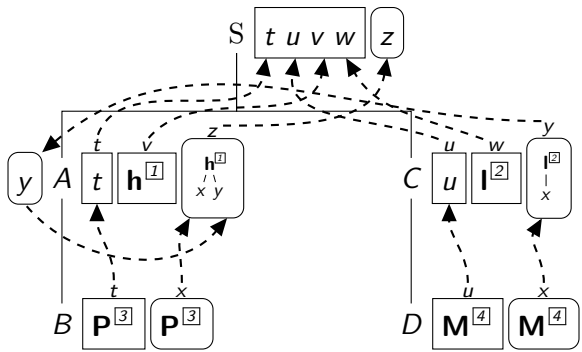
A (LCFRS,sDCP)-hybrid grammar.

$$
\begin{array}{llll}
\langle\ \mathrm{S}(tuvw) & \to\ \mathrm{A}(t,v)\ \mathrm{C}(u,w) & , & \mathrm{S}(\varepsilon;z) \to\ \mathrm{A}(y;z)\ \mathrm{C}(\varepsilon;y)\ \rangle \\
\langle\ \mathrm{A}(t,\textbf{helpen}^{\boxed{1}}) & \to\ \mathrm{B}(t) & , & \mathrm{A}(y;\textbf{helpen}^{\boxed{1}}(x,y)) \to\ \mathrm{B}(\varepsilon;x)\ \rangle \\
\langle\ \mathrm{B}(\textbf{Piet}^{\boxed{1}}) & \to\ \varepsilon & , & \mathrm{B}(\varepsilon;\textbf{Piet}^{\boxed{1}}) \to\ \varepsilon\ \rangle \\
\langle\ \mathrm{C}(u,\textbf{lezen}^{\boxed{1}}) & \to\ \mathrm{D}(u) & , & \mathrm{C}(\varepsilon;\textbf{lezen}^{\boxed{1}}(x)) \to\ \mathrm{D}(\varepsilon;x) \\
\langle\ \mathrm{D}(\textbf{Marie}^{\boxed{1}}) & \to\ \varepsilon & , & \mathrm{D}(\varepsilon;\textbf{Marie}^{\boxed{1}}) \to\ \varepsilon\ \rangle
\end{array}
$$

Derivation of a (LCFRS,sDCP)-hybrid grammar.

$$
\begin{array}{lll}
 & \langle\ \mathrm{S}(\textbf{Piet}^{\boxed{1}}\textbf{Marie}^{\boxed{2}}\textbf{helpen}^{\boxed{3}}\textbf{lezen}^{\boxed{4}}) & ,\ \mathrm{S}(\varepsilon;\textbf{helpen}^{\boxed{3}}(\textbf{Piet}^{\boxed{1}},\textbf{lezen}^{\boxed{4}}(\textbf{Marie}^{\boxed{2}})))\ \rangle \\
\Rightarrow & \langle\ \mathrm{A}(\textbf{Piet}^{\boxed{1}},\textbf{helpen}^{\boxed{3}})\ \mathrm{C}(\textbf{Marie}^{\boxed{2}},\textbf{lezen}^{\boxed{4}}) & ,\ \mathrm{A}(\xi;\textbf{helpen}^{\boxed{3}}(\textbf{Piet}^{\boxed{1}},\xi))\ \mathrm{C}(\varepsilon;\textbf{lezen}^{\boxed{4}}(\textbf{Marie}^{\boxed{2}}))\ \rangle \\
\Rightarrow & \langle\ \mathrm{B}(\textbf{Piet}^{\boxed{1}})\ \mathrm{C}(\textbf{Marie}^{\boxed{2}},\textbf{lezen}^{\boxed{4}}) & ,\ \mathrm{B}(\varepsilon;\textbf{Piet}^{\boxed{1}})\ \mathrm{C}(\varepsilon;\textbf{lezen}^{\boxed{4}}(\textbf{Marie}^{\boxed{2}}))\ \rangle \\
\Rightarrow & \langle\ \mathrm{C}(\textbf{Marie}^{\boxed{2}},\textbf{lezen}^{\boxed{4}}) & ,\ \mathrm{C}(\varepsilon;\textbf{lezen}^{\boxed{4}}(\textbf{Marie}^{\boxed{2}}))\ \rangle \\
\Rightarrow & \langle\ \mathrm{D}(\textbf{Marie}^{\boxed{2}}) & ,\ \mathrm{D}(\varepsilon;\textbf{Marie}^{\boxed{2}})\ \rangle \\
\Rightarrow & \langle\ \varepsilon & ,\ \varepsilon\ \rangle
\end{array}
$$

A derivation tree of the example hybrid grammar.



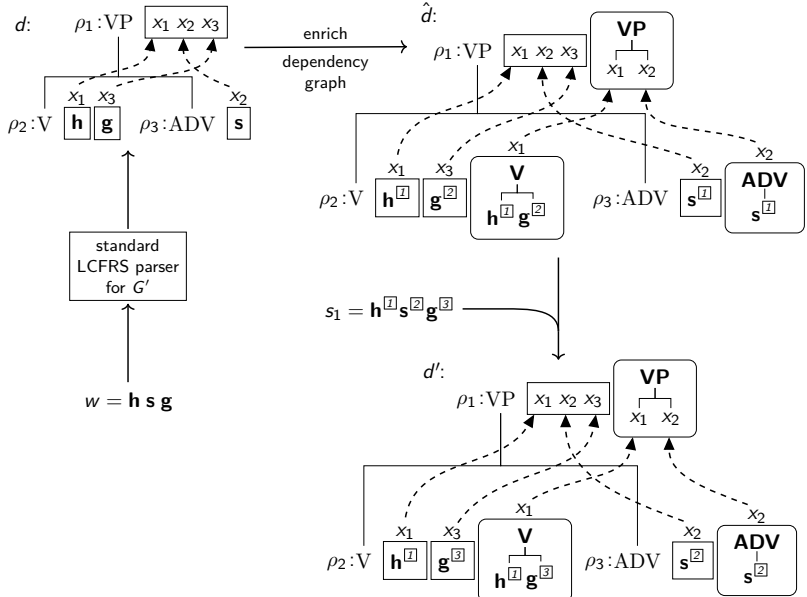
Parsing a string with probabilistic hybrid grammars [Gebhardt et al., 2017]

Input: probabilistic hybrid grammar (G, p) over Σ and Δ
$w = w_1 \cdots w_n \in \Sigma^*$

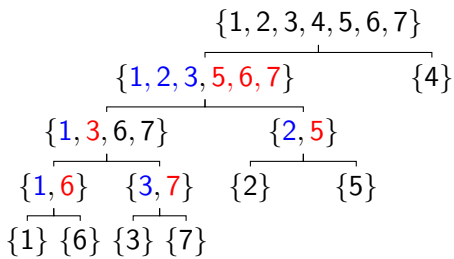
Output: hybrid tree h with $\mathrm{str}(h) = w$ and with most probable derivation

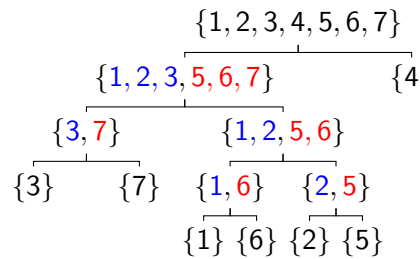
1: extract the first component of G; resulting in a probabilistic LCFRS G'
2: parse w according to G' with any standard LCFRS parser; resulting in a most likely derivation tree d
3: enrich the dependency graph of d by the arguments which correspond to the second components of the applied hybrid grammar rules; resulting in the intermediate structure $\hat{d}$
4: traverse $\hat{d}$ and create the linear order on a subset of its positions according to the order on the corresponding positions of w; resulting in derivation tree d'
5: **return** hybrid tree h corresponding to $\tau(d')$

Example: Parsing a string with probabilistic hybrid grammars [Gebhardt et al., 2017]



Two examples of recursive partitionings of a string of length 7.





Induction of a LCFRS from a string and a recursive partitioning.

Input: a string $w = w_1 \cdots w_n \in \Sigma^*$
a recursive partitioning π of w

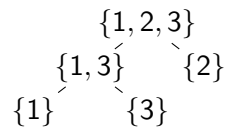
Output: a simple LCFRS G that parses w according to π; fanout of G equals fanout of π

function INDUCE_LCFRS(w,π)
  $P \leftarrow \emptyset$ ▷ set of LCFRS rules
  for $p \in \mathrm{pos}(\pi)$ **do**
    $j \leftarrow$ number of children of p in π
    $J_0 \leftarrow \pi(p),\ J_1 \leftarrow \pi(p1),\ \ldots,\ J_j \leftarrow \pi(pj)$
    if $J_0 = \{i\}$ for some $i \in [n]$ **then**
      $P \leftarrow P \cup \{(\!|\{i\}|\!)(w_i) \rightarrow \varepsilon\}$
    else
      for $\ell \in [j]_0$ **do** $\langle J_{\ell,1}, \ldots, J_{\ell,k_\ell} \rangle \leftarrow \mathrm{spans}(J_\ell)$
      for $q \in [k_0]$ **do**
        let r and $\ell_1, \ldots, \ell_r \in [j]$ and $q_1 \in [k_{\ell_1}], \ldots, q_r \in [k_{\ell_r}]$ be such that
        $J_{0,q} = J_{\ell_1,q_1} \cup \cdots \cup J_{\ell_r,q_r}$ and $i \in J_{\ell_t,q_t}, i' \in J_{\ell_{t+1},q_{t+1}}$ implies $i < i'$
        $s_q \leftarrow x_{q_1}^{(l_1)} \ldots x_{q_r}^{(\ell_r)}$
      $P \leftarrow P \cup \{(\!|J_0|\!)(s_{1,k_0}) \rightarrow (\!|J_1|\!)(x_{1,k_1}^{(1)})\ \ldots\ (\!|J_j|\!)(x_{1,k_j}^{(j)})\}$
  return $G = (N, (\!|[n]|\!), \Sigma, P)$, where $N = \{(\!|J|\!) \mid J \text{ is label in } \pi\}$

Induction of a LCFRS from a string and a recursive partitioning (2)

string: **h s g**

recursive partitioning:



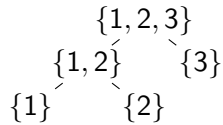
induced rules:

$$\begin{aligned}
(\!|\{1,2,3\}|\!)(x_1 x_3 x_2) &\rightarrow (\!|\{1,3\}|\!)(x_1, x_2)\ (\!|\{2\}|\!)(x_3) \\
(\!|\{1,3\}|\!)(x_1, x_2) &\rightarrow (\!|\{1\}|\!)(x_1)\ (\!|\{3\}|\!)(x_2) \\
(\!|\{1\}|\!)(\mathbf{h}) \rightarrow \varepsilon \quad (\!|\{2\}|\!)(\mathbf{s}) &\rightarrow \varepsilon \quad (\!|\{3\}|\!)(\mathbf{g}) \rightarrow \varepsilon
\end{aligned}$$

Induction of a LCFRS from a string and a recursive partitioning (3)

string: **h s g**

recursive partitioning:



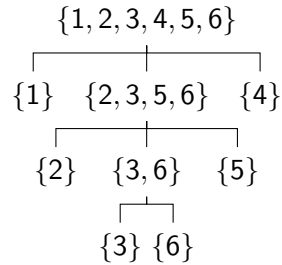
induced rules:

$$(\!|\{1,2,3\}|\!)(x_1x_2) \rightarrow (\!|\{1,2\}|\!)(x_1)\ (\!|\{3\}|\!)(x_2)$$
$$(\!|\{1,2\}|\!)(x_1x_2) \rightarrow (\!|\{1\}|\!)(x_1)\ (\!|\{2\}|\!)(x_2)$$
$$(\!|\{1\}|\!)(\mathbf{h}) \rightarrow \varepsilon \quad (\!|\{2\}|\!)(\mathbf{s}) \rightarrow \varepsilon \quad (\!|\{3\}|\!)(\mathbf{g}) \rightarrow \varepsilon$$

Induction of a LCFRS from a string and a recursive partitioning (4)

string: $\mathbf{Jan}_1\ \mathbf{Piet}_2\ \mathbf{Marie}_3\ \mathbf{zag}_4\ \mathbf{helpen}_5\ \mathbf{lezen}_6$

recursive partitioning:



induced rules:

$$(\!|\{1,2,3,4,5,6\}|\!)(x_1^{(1)}x_1^{(2)}x_1^{(4)}x_2^{(2)}) \to (\!|\{1\}|\!)(x_1^{(1)})\ \ (\!|\{2,3,5,6\}|\!)(x_1^{(2)},x_2^{(2)})\ \ (\!|\{4\}|\!)(x_1^{(4)})$$

$$(\!|\{2,3,5,6\}|\!)(x_1^{(1)}x_1^{(2)},x_1^{(3)}x_2^{(2)}) \to (\!|\{2\}|\!)(x_1^{(1)})\ \ (\!|\{3,6\}|\!)(x_1^{(2)},x_2^{(2)})\ \ (\!|\{5\}|\!)(x_1^{(3)})$$

$$(\!|\{3,6\}|\!)(x_1^{(1)},x_1^{(2)}) \to (\!|\{3\}|\!)(x_1^{(1)})\ \ (\!|\{6\}|\!)(x_1^{(2)})$$

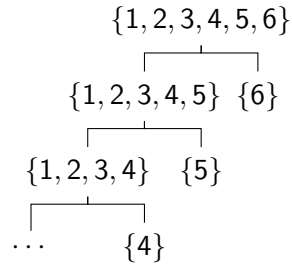
$$(\!|\{6\}|\!)(\mathbf{lezen}) \to \varepsilon \quad (\!|\{5\}|\!)(\mathbf{helpen}) \to \varepsilon \quad (\!|\{4\}|\!)(\mathbf{zag}) \to \varepsilon$$

$$(\!|\{3\}|\!)(\mathbf{Marie}) \to \varepsilon \quad (\!|\{2\}|\!)(\mathbf{Piet}) \to \varepsilon \quad (\!|\{1\}|\!)(\mathbf{Jan}) \to \varepsilon$$

Induction of a LCFRS from a string and a recursive partitioning (5)

string: $\mathbf{Jan}_1\ \mathbf{Piet}_2\ \mathbf{Marie}_3\ \mathbf{zag}_4\ \mathbf{helpen}_5\ \mathbf{lezen}_6$

recursive partitioning:



induced rules:

$$(\!|\{1,2,3,4,5,6\}|\!)(x_1^{(1)}x_1^{(2)}) \to (\!|\{1,2,3,4,5\}|\!)(x_1^{(1)})\ \ (\!|\{6\}|\!)(x_1^{(2)})$$

$$(\!|\{1,2,3,4,5\}|\!)(x_1^{(1)}x_1^{(2)}) \to (\!|\{1,2,3,4\}|\!)(x_1^{(1)})\ \ (\!|\{5\}|\!)(x_1^{(2)})$$

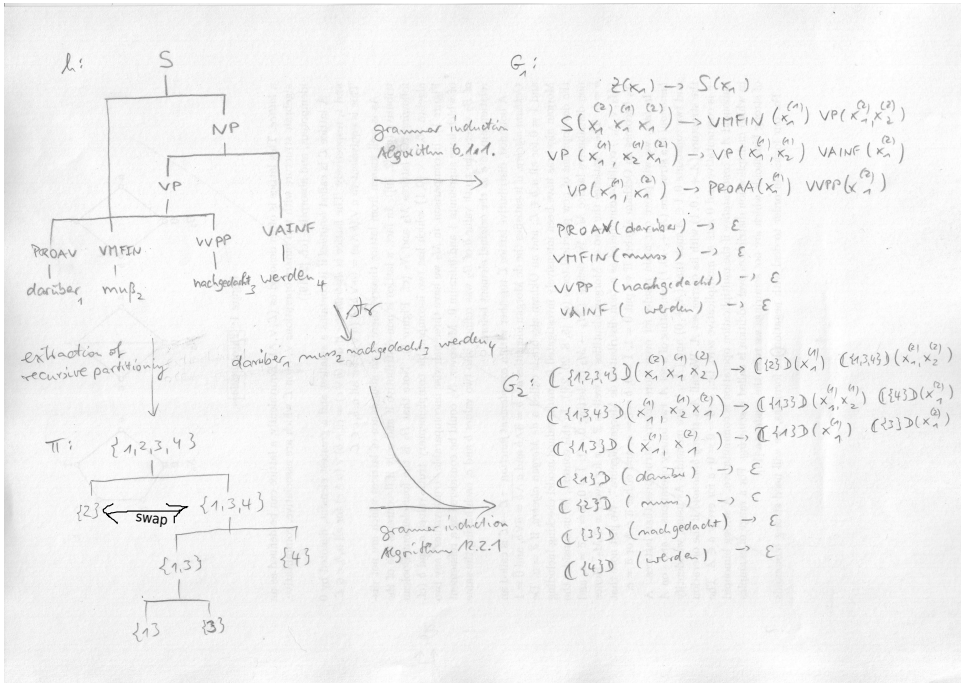
$$(\!|\{1,2,3,4\}|\!)(x_1^{(1)}x_1^{(2)}) \to (\!|\{1,2,3\}|\!)(x_1^{(1)})\ \ (\!|\{4\}|\!)(x_1^{(2)})$$

$$\ldots$$

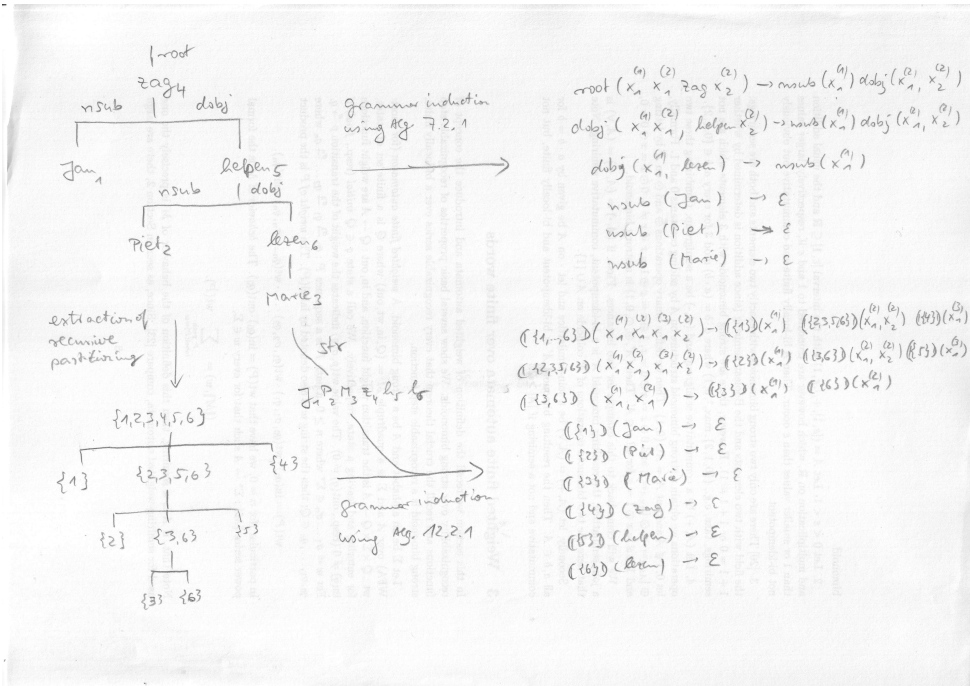
$$(\!|\{6\}|\!)(\mathbf{lezen}) \to \varepsilon \quad (\!|\{5\}|\!)(\mathbf{helpen}) \to \varepsilon \quad (\!|\{4\}|\!)(\mathbf{zag}) \to \varepsilon$$

$$(\!|\{3\}|\!)(\mathbf{Marie}) \to \varepsilon \quad (\!|\{2\}|\!)(\mathbf{Piet}) \to \varepsilon \quad (\!|\{1\}|\!)(\mathbf{Jan}) \to \varepsilon$$

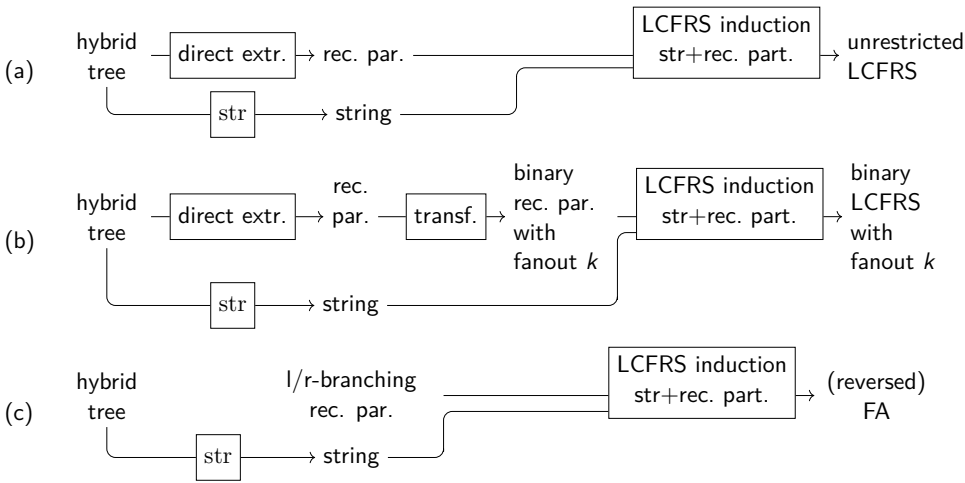
Comparison LCFRS induction: direct vs. recursive partitioning (1)



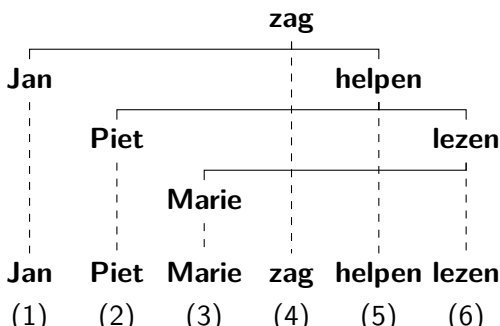
Comparison LCFRS induction: direct vs. recursive partitioning (2)

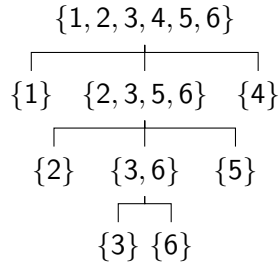


Three pipelines for LCFRS induction



A dependency structure and the recursive partitioning extracted from it





Extraction of recursive partitioning from hybrid tree

Input: a hybrid tree $h = (\xi, U, \leq)$ with $U = \{p_1, \ldots, p_n\}$ and $n > 0$ where $p_i \leq p_{i+1}$ for each $i \in [n-1]$

Output: a recursive partitioning of $\mathrm{str}(h)$

1: **function** EXTRACT_RECURSIVE_PARTITIONING(h)
2: **return** REC_PAR(ε)
3: **function** REC_PAR(p) ▷ $p \in \mathrm{pos}(\xi)$
4: $v \leftarrow \varepsilon$ ▷ $v \in (T_{\mathcal{P}([n])})^*$
5: **if** $p \in U$ **then** $v \leftarrow \{i\}$ if $p = p_i$
6: **for** $p' \in \mathrm{children}(p)$ **do**
7: $v \leftarrow v \cdot \mathrm{REC\_PAR}(p')$ ▷ concatenation of strings v and REC_PAR(p')
8: **if** $|v| \leq 1$ **then return** v
9: **else**
10: $V \leftarrow \bigcup_{j=1}^{|v|} v(j)(\varepsilon)$
11: sort v such that $\min(v(j)(\varepsilon)) < \min(v(j+1)(\varepsilon))$
12: **return** $V(v(1), \ldots, v(|v|))$

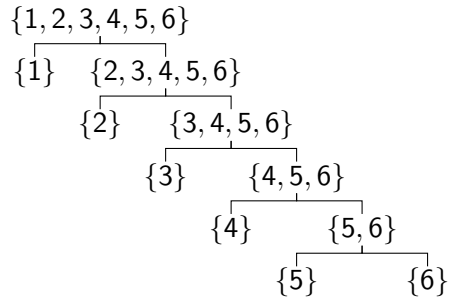
Transformation of recursive partitioning

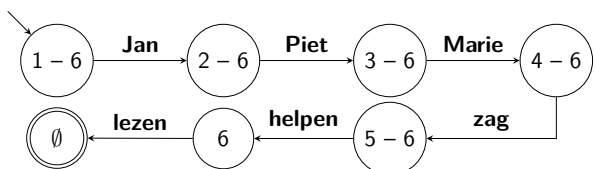
Input: a recursive partitioning π of a string of length n
an integer $k \geq 1$

Output: a binary recursive partitioning π' of fanout no greater than k

function $\textsc{Transform}(\pi = J(t_1, \ldots, t_m))$
  if $|J| = 1$ **then**
    return J
  breadth-first search p in $\mathrm{pos}(\pi) \setminus \{\varepsilon\}$ such that $\pi(p)$ and $J \setminus \pi(p)$ have fanout $\leq k$
  $t \leftarrow \textsc{Filter}(\pi(p), \pi)$
  return $J(\textsc{Transform}(\pi|_p), \textsc{Transform}(t))$
function $\textsc{Filter}(J', \pi = J(t_1, \ldots, t_m))$ ▷ $J' \subseteq [n]$, $\pi \in (T_{\mathcal{P}([n])})^*$
  $F \leftarrow J \setminus J'$
  if $|F| = 1$ **then** **return** F
  else if $|F| = 0$ **then return** ε
  else
    $s \leftarrow \textsc{Filter}(J', t_1) \cdot \ldots \cdot \textsc{Filter}(J', t_m)$
    if $|s| = 1$ **then** **return** s
    else **return** $F(s(1), \ldots, s(|s|))$

A right-branching recursive partitioning and an FA





Induction of sDCP from phrase structure tree and rec. part. I

Input: a phrase structure tree $h = (\xi, \text{leaves}(\xi), \preceq)$ with $\text{leaves}(\xi) = \{p_1, \ldots, p_n\}$
where $p_i \preceq p_{i+1}$ for each $i \in [n-1]$
a recursive partitioning π of $\text{str}(h)$

Output: a sDCP G that generates ξ according to π

---

function CONSTRUCT_sDCP($(\xi, \text{leaves}(\xi), \preceq), \pi$)
  $P \leftarrow \emptyset$ ▷ set of sDCP rules
  for $p \in \text{pos}(\pi)$ **do**
    $m \leftarrow$ number of children of p in π
    $J_0 \leftarrow \pi(p)$, $J_1 \leftarrow \pi(p1)$, $\ldots$, $J_m \leftarrow \pi(pm)$
    $\langle I_1^{(0)}, \ldots, I_{k_0'}^{(0)} \rangle \leftarrow \text{gspans}(C(J_0))$
    $\langle O_1^{(i)}, \ldots, O_{k_i}^{(i)} \rangle \leftarrow \text{gspans}(C(J_i))$ for each $i \in [m]$
    for each $q \in [k_0']$ **do**
      $r \leftarrow \min_{\leq_\ell}(I_q^{(0)})$ ▷ r is the root of $I_q^{(0)}$
      $\xi_q \leftarrow$ CONSTRTREE($r, (O_j^{(i)} \mid i \in [m], j \in [k_i])$)
    $P \leftarrow P \cup \{(\!|J_0|\!)(\varepsilon; \xi_{1,k_0'}) \rightarrow (\!|J_1|\!)(\varepsilon; x_{1,k_1}^{(1)}) \ldots (\!|J_m|\!)(\varepsilon; x_{1,k_m}^{(m)})\}$
  return $G = (N, (\!|[n]|\!), \Sigma, P)$, where $N = \{(\!|J|\!) \mid J \text{ label in } \pi\}$

Induction of sDCP from phrase structure tree and rec. part. II

13: **function** ConstrTree(r, $(O_j^{(i)} \mid i \in [m], j \in [k_i])$)
$\triangleright$ $r \in \mathrm{pos}(\xi)$, $(O_j^{(i)} \mid i \in [m], j \in [k_i])$ family of subsets of $\mathrm{pos}(\xi)$
14: **if** $r \in O_j^{(i)}$ for some $i \in [m]$, $j \in [k_i]$ **then**
15: **return** $x_j^{(i)}$
16: **else**
17: let $\mathrm{children}(r) = (r_1, \ldots, r_\ell)$
18: **for** $q \in [\ell]$ **do**
19: $\zeta_q \leftarrow$ ConstrTree$(r_q, (O_j^{(i)} \mid i \in [m], j \in [k_i]))$
20: **return** $\sigma(\zeta_1, \ldots, \zeta_\ell)$ where $\sigma = \xi(r)$

Induction of sDCP from dependency tree and rec. part. I

Input: a dependency tree $h = (\xi, \mathrm{lab}, U, \preceq)$ with $U = \{p_1, \ldots, p_n\}$; $p_i \preceq p_{i+1}$
a recursive partitioning π of $\mathrm{str}(h)$

Output: a sDCP G that generates ξ according to π

---

function CONSTRUCT_sDCP($(\xi, \mathrm{lab}, U, \preceq), \pi$)
  $P \leftarrow \emptyset$ ▷ set of sDCP rules
  for $p \in \mathrm{pos}(\pi)$ **do**
    $m \leftarrow$ number of children of p in π
    $J_0 \leftarrow \pi(p)$, $J_1 \leftarrow \pi(p1), \ldots, J_m \leftarrow \pi(pm)$
    $\langle I_1^{(0)}, \ldots, I_{k_0'}^{(0)} \rangle \leftarrow$ gspans($\top(\Pi(J_0))$) ▷ I: inside attributes
    $\langle O_1^{(0)}, \ldots, O_{k_0}^{(0)} \rangle \leftarrow$ gspans($\bot(\Pi(J_0))$) ▷ O: outside attributes
    $\langle O_1^{(i)}, \ldots, O_{k_i}^{(i)} \rangle \leftarrow$ gspans($\top(\Pi(J_i))$) for each $i \in [m]$
    $\langle I_1^{(i)}, \ldots, I_{k_i'}^{(i)} \rangle \leftarrow$ gspans($\bot(\Pi(J_i))$) for each $i \in [m]$ ▷ note: each $I_q^{(i)}$ and $O_q^{(i)}$ is a singleton
    for each $\ell \in [m]_0$ and $q \in [k_\ell']$ **do**
      let $I_q^{(\ell)} = \{r\}$
      $\xi_q^{(\ell)} \leftarrow$ CONSTRTREE($r, (O_j^{(i)} \mid i \in [m]_0, j \in [k_i])$)
    $P \leftarrow P \cup \{(\!|J_0|\!)(x_{1,k_0}^{(0)}; \xi_{1,k_0'}^{(0)}) \rightarrow (\!|J_1|\!)(\xi_{1,k_1'}^{(1)}; x_{1,k_1}^{(1)}) \ \ldots \ (\!|J_m|\!)(\xi_{1,k_m'}^{(m)}; x_{1,k_m}^{(m)})\}$
  return $G = (N, (\!|[n]|\!), \Sigma, P)$, where $N = \{(\!|J|\!) \mid J \text{ label in } \pi\}$

Induction of sDCP from dependency tree and rec. part. II

15: **function** ConstrTree(r, $(O_j^{(i)} \mid i \in [m]_0, j \in [k_i])$)
$\triangleright$ $r \in \mathrm{pos}(\xi)$, $(O_j^{(i)} \mid i \in [m]_0, j \in [k_i])$ family of subsets of $\mathrm{pos}(\xi)$
16: **if** $r \in O_j^{(i)}$ for some $i \in [m]_0$, $j \in [k_i]$ **then**
17: **return** $x_j^{(i)}$
18: **else**
19: let $\mathrm{children}(r) = (r_1, \ldots, r_\ell)$
20: **for** $q \in [\ell]$ **do**
21: $\zeta_q \leftarrow$ ConstrTree$(r_q, (O_j^{(i)} \mid i \in [m]_0, j \in [k_i]))$
22: **return** $\sigma(\zeta_1, \ldots, \zeta_\ell)$ where $\sigma = \xi(r)$

grammar induction from one hybrid tree:

hybrid tree
$h = (\xi, U, \preceq)$

grammar induction from one hybrid tree:

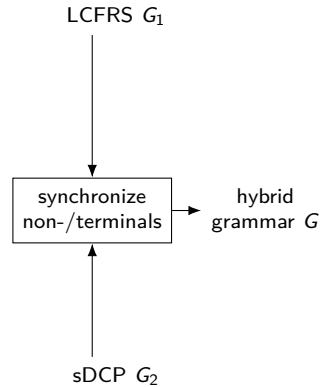
hybrid tree
$h = (\xi, U, \preceq)$

hybrid
grammar G

$L(G) = \{h\}$

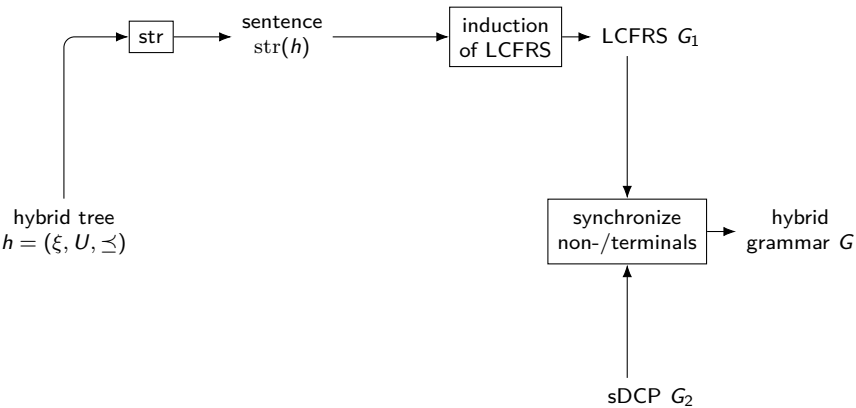
grammar induction from one hybrid tree:

hybrid tree
$h = (\xi, U, \preceq)$



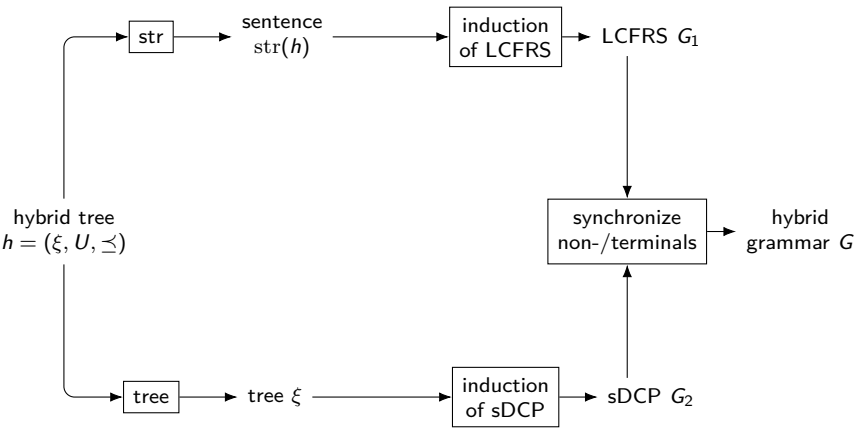
$L(G) = \{h\}$

grammar induction from one hybrid tree:



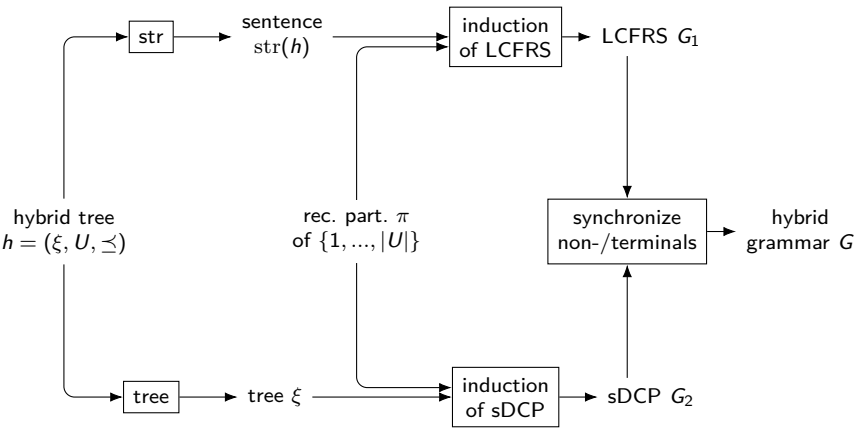
$L(G) = \{h\}$

grammar induction from one hybrid tree:



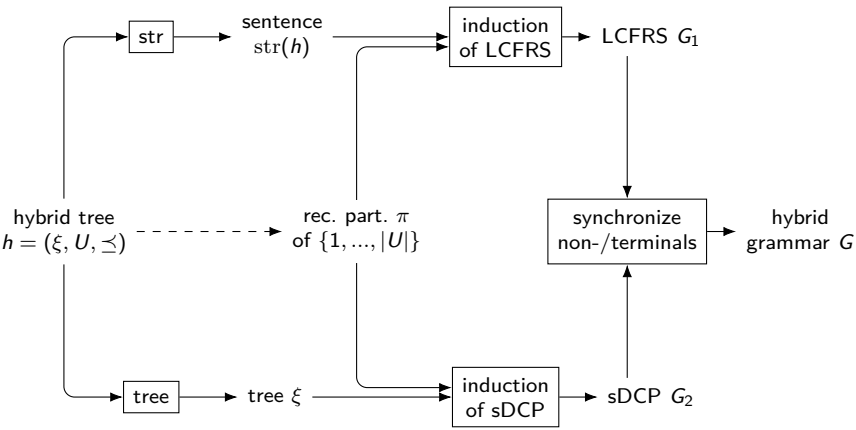
$L(G) = \{h\}$

grammar induction from one hybrid tree:



$L(G) = \{h\}$
parsing of $\mathrm{str}(h)$ according to π

grammar induction from one hybrid tree:



$L(G) = \{h\}$
parsing of $\mathrm{str}(h)$ according to π

$$
\begin{aligned}
\langle\ (\!|\{1,2,3\}|\!)(x_1x_3x_2) &\ \rightarrow\ (\!|\{1,3\}|\!)(x_1,x_2)\ (\!|\{2\}|\!)(x_3)\ , \\
(\!|\{1,2,3\}|\!)(\varepsilon;\mathbf{VP}(x_1,x_2)) &\ \rightarrow\ (\!|\{1,3\}|\!)(\varepsilon;x_1)\ (\!|\{2\}|\!)(\varepsilon;x_2)\ \rangle
\end{aligned}
$$

$$
\begin{aligned}
\langle\ (\!|\{1,3\}|\!)(x_1,x_2) &\ \rightarrow\ (\!|\{1\}|\!)(x_1)\ (\!|\{3\}|\!)(x_2)\ , \\
(\!|\{1,3\}|\!)(\varepsilon;\mathbf{V}(x_1,x_2)) &\ \rightarrow\ (\!|\{1\}|\!)(\varepsilon;x_1)\ (\!|\{3\}|\!)(\varepsilon;x_2)\ \rangle
\end{aligned}
$$

$$
\langle\ (\!|\{1\}|\!)(\mathbf{h}^{\boxed{I}})\ \rightarrow\ \varepsilon\quad ,\quad (\!|\{1\}|\!)(\varepsilon;\mathbf{h}^{\boxed{I}})\ \rightarrow\ \varepsilon\ \rangle
$$

$$
\langle\ (\!|\{2\}|\!)(\mathbf{s}^{\boxed{I}})\ \rightarrow\ \varepsilon\quad ,\quad (\!|\{2\}|\!)(\varepsilon;\mathbf{ADV}(\mathbf{s}^{\boxed{I}}))\ \rightarrow\ \varepsilon\ \rangle
$$

$$
\langle\ (\!|\{3\}|\!)(\mathbf{g}^{\boxed{I}})\ \rightarrow\ \varepsilon\quad ,\quad (\!|\{3\}|\!)(\varepsilon;\mathbf{g}^{\boxed{I}})\ \rightarrow\ \varepsilon\ \rangle
$$

$$
\begin{array}{rcl}
\langle\ (\!|\{1,2,3\}|\!)(x_1 x_2) & \to & (\!|\{1,2\}|\!)(x_1)\ (\!|\{3\}|\!)(x_2)\ , \\
(\!|\{1,2,3\}|\!)(\varepsilon; \mathbf{VP}(\mathbf{V}(x_1, x_3), x_2)) & \to & (\!|\{1,2\}|\!)(\varepsilon; x_1, x_2)\ (\!|\{3\}|\!)(\varepsilon; x_3)\ \rangle \\
\\
\langle\ (\!|\{1,2\}|\!)(x_1 x_2) & \to & (\!|\{1\}|\!)(x_1)\ (\!|\{2\}|\!)(x_2)\ , \\
(\!|\{1,2\}|\!)(\varepsilon; x_1, x_2) & \to & (\!|\{1\}|\!)(\varepsilon; x_1)\ (\!|\{2\}|\!)(\varepsilon; x_2)\ \rangle \\
\\
\langle\ (\!|\{1\}|\!)(\mathbf{h}^{\boxed{1}}) \to \varepsilon & , & (\!|\{1\}|\!)(\varepsilon; \mathbf{h}^{\boxed{1}}) \to \varepsilon\ \rangle \\
\\
\langle\ (\!|\{2\}|\!)(\mathbf{s}^{\boxed{1}}) \to \varepsilon & , & (\!|\{2\}|\!)(\varepsilon; \mathbf{ADV}(\mathbf{s}^{\boxed{1}})) \to \varepsilon\ \rangle \\
\\
\langle\ (\!|\{3\}|\!)(\mathbf{g}^{\boxed{1}}) \to \varepsilon & , & (\!|\{3\}|\!)(\varepsilon; \mathbf{g}^{\boxed{1}}) \to \varepsilon\ \rangle
\end{array}
$$

$$
\begin{array}{rcl}
\langle\ (\!|\{2,3,5,6\}|\!)(x_1^{(1)}x_1^{(2)}, x_2^{(2)}x_2^{(1)}) & \to & (\!|\{2,6\}|\!)(x_1^{(1)}, x_2^{(1)})\ \ (\!|\{3,5\}|\!)(x_1^{(2)}, x_2^{(2)}) \\
(\!|\{2,3,5,6\}|\!)(\varepsilon; x_2^{(2)}) & \to & (\!|\{2,6\}|\!)(x_1^{(2)};\ x_1^{(1)}, x_2^{(1)}) \\
 & & \ \ (\!|\{3,5\}|\!)(x_1^{(1)}, x_2^{(1)}; x_1^{(2)}, x_2^{(2)})\ \rangle
\end{array}
$$

$$
\begin{array}{rcl}
\langle\ (\!|\{2,6\}|\!)(x_1^{(1)}, x_1^{(2)}) & \to & (\!|\{2\}|\!)(x_1^{(1)})\ \ (\!|\{6\}|\!)(x_1^{(2)}) \\
(\!|\{2,6\}|\!)(x_1^{(0)};\ x_1^{(1)}, x_1^{(2)}) & \to & (\!|\{2\}|\!)(\varepsilon;\ x_1^{(1)})\ \ (\!|\{6\}|\!)(x_1^{(0)};\ x_1^{(2)})\ \rangle
\end{array}
$$

$$
\begin{array}{rcl}
\langle\ (\!|\{3,5\}|\!)(x_1^{(1)}, x_1^{(2)}) & \to & (\!|\{3\}|\!)(x_1^{(1)})\ \ (\!|\{5\}|\!)(x_1^{(2)}) \\
(\!|\{3,5\}|\!)(x_1^{(0)}, x_2^{(0)};\ x_1^{(1)}, x_1^{(2)}) & \to & (\!|\{3\}|\!)(\varepsilon;\ x_1^{(1)})\ \ (\!|\{5\}|\!)(x_1^{(0)}, x_2^{(0)};\ x_1^{(2)})\ \rangle
\end{array}
$$

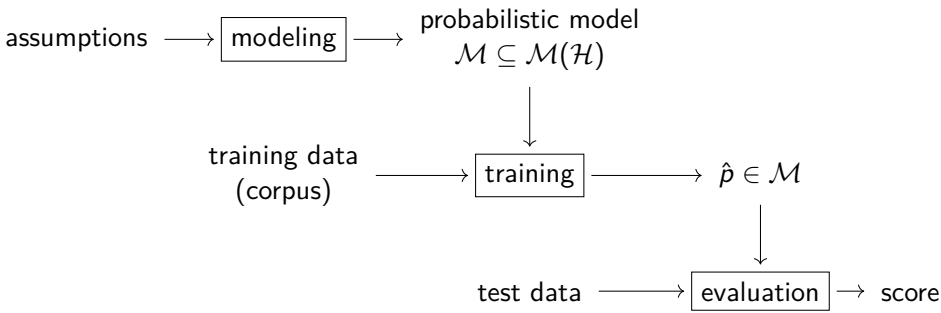
$$
\langle\ (\!|\{5\}|\!)(\mathbf{helpen}^{\boxed{1}}) \to \varepsilon \quad , \quad (\!|\{5\}|\!)(x_1^{(0)}, x_2^{(0)};\ \mathbf{helpen}^{\boxed{1}}(x_1^{(0)}, x_2^{(0)})) \to \varepsilon\ \rangle
$$

$$
(\!|\{6\}|\!)(\mathbf{lezen}^{\boxed{1}}) \to \varepsilon \quad , \quad (\!|\{6\}|\!)(x_1^{(0)};\ \mathbf{lezen}^{\boxed{1}}(x_1^{(0)})) \to \varepsilon\ \rangle
$$

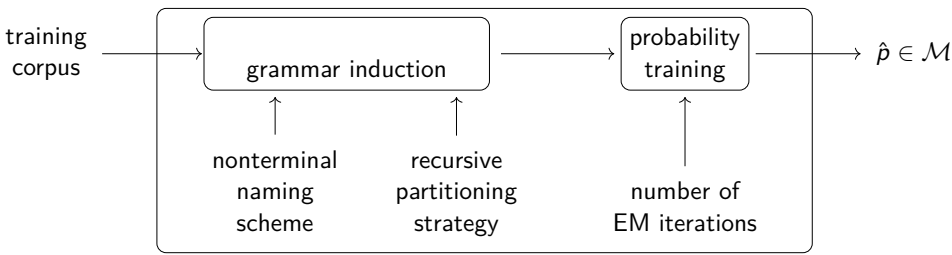
$$
(\!|\{3\}|\!)(\mathbf{Marie}^{\boxed{1}}) \to \varepsilon \quad , \quad (\!|\{3\}|\!)(\varepsilon;\ \mathbf{Marie}^{\boxed{1}}) \to \varepsilon\ \rangle
$$

$$
(\!|\{2\}|\!)(\mathbf{Piet}^{\boxed{1}}) \to \varepsilon \quad , \quad (\!|\{2\}|\!)(\varepsilon;\ \mathbf{Piet}^{\boxed{1}}) \to \varepsilon
$$

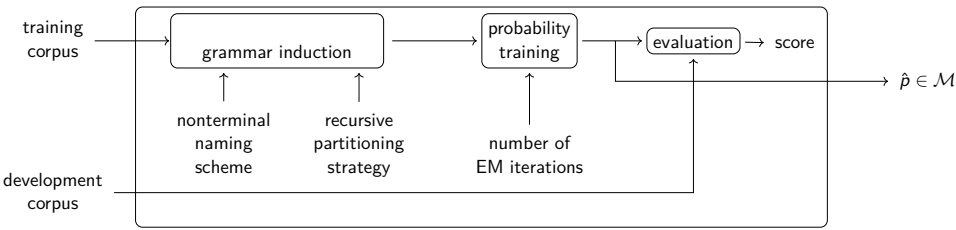
A (too) simple pipeline for modelling, training, and testing



A more detailed look on training unveils various hyperparameters



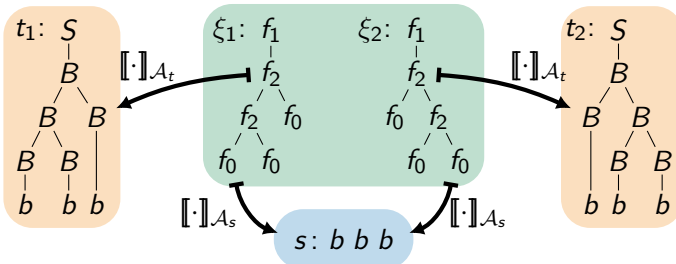
A training pipeline with a development set



extraction	arg. lab.	nont.	rules	f_{max}	f_{avg}	fail	UASp	LASp	UAS	LAS	LA	time
child labeling												
direct	P+D	4,739	27,042	7	1.10	693	51.7	40.7	52.2	40.8	42.6	253
$k = 1$	P+D	13,178	35,071	1	1.00	202	75.9	68.7	77.0	69.1	73.3	288
$k = 2$	P+D	11,156	32,231	2	1.17	195	76.5	69.6	77.7	70.1	74.2	355
r-branch	P+D	42,577	79,648	1	1.00	775	45.5	33.1	45.7	32.8	34.7	49
l-branch	P+D	40,100	75,321	1	1.00	768	45.8	33.4	46.0	33.2	35.0	45
direct	POS	675	19,276	7	1.24	303	68.7	51.5	69.3	50.0	55.4	300
$k = 1$	POS	3,464	15,826	1	1.00	30	**81.7**	65.5	**82.5**	63.5	70.6	244
$k = 2$	POS	2,099	13,347	2	1.40	35	81.6	65.2	82.4	63.3	70.5	410
r-branch	POS	19,804	51,733	1	1.00	372	62.7	46.4	62.7	44.7	50.6	222
l-branch	POS	17,240	45,883	1	1.00	342	63.7	47.4	63.9	45.6	51.4	197
direct	DEP	2,505	19,511	7	1.13	3	78.5	72.2	78.9	71.6	78.6	484
$k = 1$	DEP	8,059	22,613	1	1.00	1	78.5	71.7	79.5	71.7	79.0	608
$k = 2$	DEP	6,651	20,314	2	1.20	1	78.7	**72.1**	79.8	**72.0**	**79.2**	971
$k = 3$	DEP	6,438	19,962	3	1.25	1	78.6	72.0	79.5	71.9	79.1	1,013
r-branch	DEP	27,653	54,360	1	1.00	2	76.0	68.4	76.3	67.5	76.1	216
l-branch	DEP	25,699	50,418	1	1.00	1	75.8	68.4	76.2	67.6	76.1	198
cascade: child labeling, $k = 1$, P+D/POS/DEP						1	**83.2**	**76.2**	**84.3**	**76.1**	**81.6**	325
LCFRS Maier and Kallmeyer [2010]						-	79.0	71.8	-	-	-	-
rparse simple		920	18,587	7	1.37	56	77.1	70.6	77.3	70.0	76.2	350
rparse ($v = 1$, $h = 3$)		40,141	61,450	7	1.10	13	78.4	72.2	78.5	71.4	79.0	778
MaltParser, unlexicalized, stacklazy						0	85.0	80.2	85.6	80.0	85.0	24

A ranked alphabet Σ and algebras $\mathcal{A}_s$ and $\mathcal{A}_t$ for CFG parsing

$\Sigma = \{f_0, f_1, f_2\}$



IRTG [Koller and Kuhlmann, 2011]

T_Σ: derivation trees

T: parse trees

S: sentences

$$f_0^{\mathcal{A}_s}() = b$$
$$f_1^{\mathcal{A}_s}(x_1) = x_1$$
$$f_2^{\mathcal{A}_s}(x_1, x_2) = x_1 x_2$$

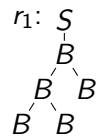
$$f_0^{\mathcal{A}_t}() = B(b)$$
$$f_1^{\mathcal{A}_t}(x_1) = S(x_1)$$
$$f_2^{\mathcal{A}_t}(x_1, x_2) = B(x_1, x_2)$$

Based on Gebhardt [2018].

Two RTGs over Σ and their valid runs on ξ_1 and ξ_2 with probabilities.

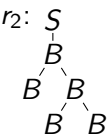
$$
\begin{aligned}
(G, p)\colon S &\to f_1(B) && \#1.0\\
B &\to f_2(B, B) && \#0.2\\
B &\to f_0() && \#0.8
\end{aligned}
$$

r_1:



$0.2^2 \cdot 0.8^3$

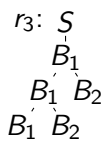
r_2:



$0.2^2 \cdot 0.8^3$

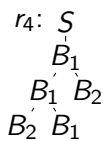
$$
\begin{aligned}
(G', p')\colon S &\to f_1(B_1) && \#1.0\\
B_1 &\to f_2(B_1, B_2) && \#0.5\\
B_1 &\to f_2(B_2, B_1) && \#0.25\\
B_1 &\to f_0() && \#0.25\\
B_2 &\to f_0() && \#1.0
\end{aligned}
$$

r_3:



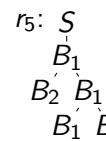
$0.5^2 \cdot 0.25^1$

r_4:



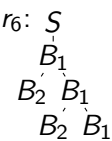
$0.5^1 \cdot 0.25^2$

r_5:



$0.5^1 \cdot 0.25^2$

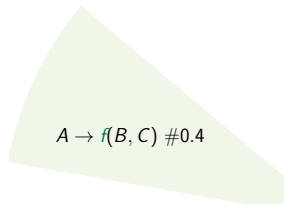
r_6:



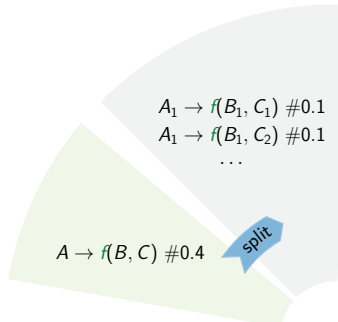
0.25^3

Based on Gebhardt [2018].

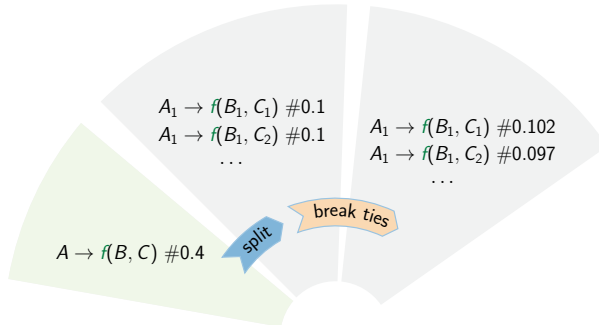
Generic grammar refinement [Gebhardt, 2018]



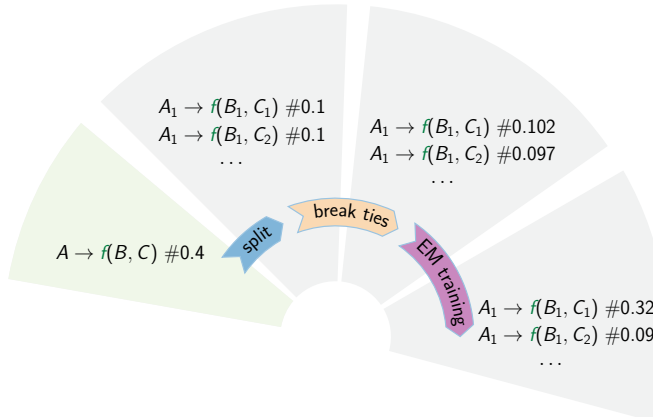
Generic grammar refinement [Gebhardt, 2018]



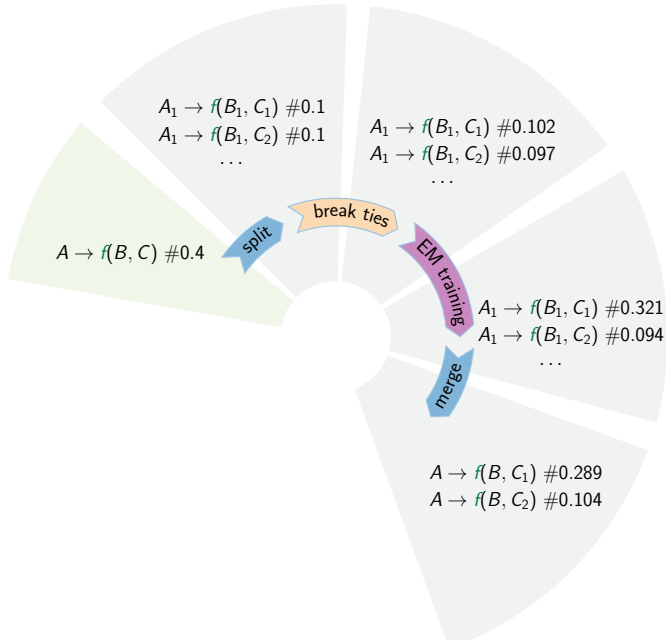
Generic grammar refinement [Gebhardt, 2018]



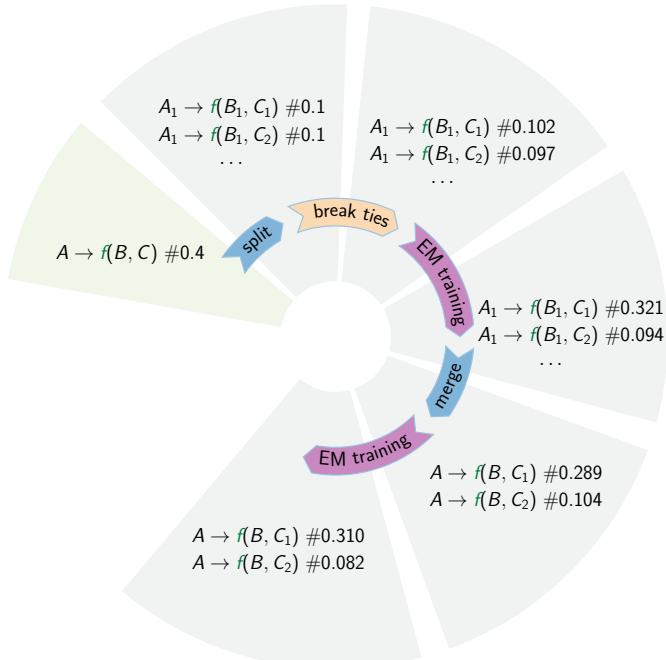
Generic grammar refinement [Gebhardt, 2018]



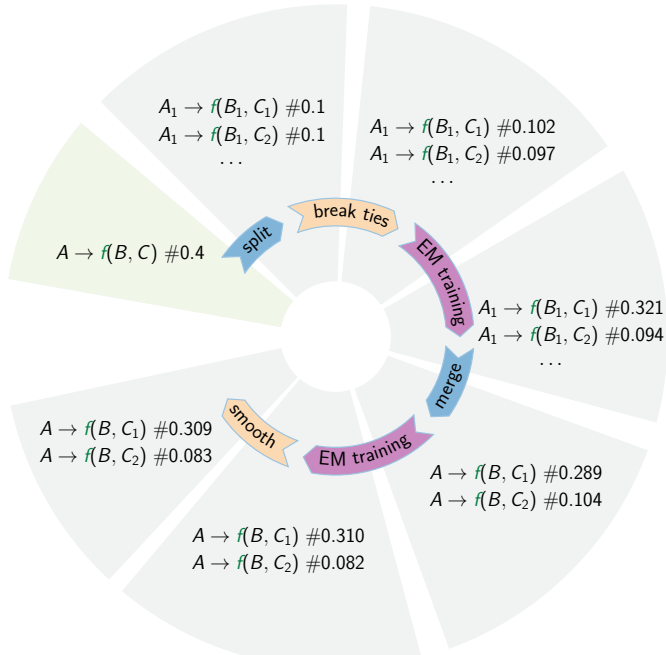
Generic grammar refinement [Gebhardt, 2018]



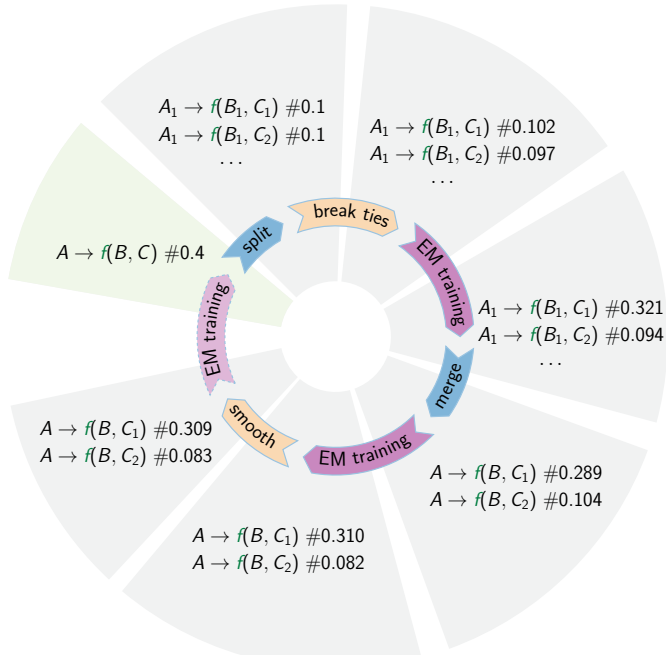
Generic grammar refinement [Gebhardt, 2018]



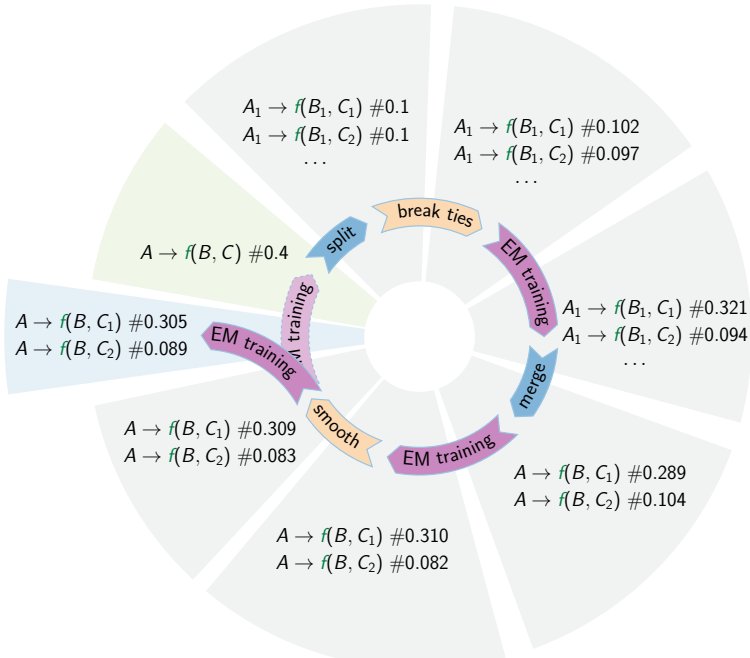
Generic grammar refinement [Gebhardt, 2018]



Generic grammar refinement [Gebhardt, 2018]



Generic grammar refinement [Gebhardt, 2018]



Parsing objectives

The probability of a parse tree is obtained by summing over the probabilities of all runs on all derivation trees for this parse tree:

$$P(t, s \mid \mathbb{G}, p) = \sum_{\substack{\xi \in T_\Sigma : \\ [\![\xi]\!]_{\mathcal{A}_t} = t \\ [\![\xi]\!]_{\mathcal{A}_s} = s}} \sum_{r \in \mathrm{runs}_G^\nu(\xi)} \prod_{\substack{\pi \in \mathrm{pos}(\xi) : \\ \mathrm{rule}_r^\pi = A \to f(B_1, \ldots, B_k)}} p(A \to f(B_1, \ldots, B_k)) \qquad \text{Exact}$$

Parsing objectives

The probability of a parse tree is obtained by summing over the probabilities of all runs on all derivation trees for this parse tree:

$$P(t, s \mid \mathbb{G}, p) = \sum_{\substack{\xi \in T_\Sigma : \\ [\![\xi]\!]_{\mathcal{A}_t} = t \\ [\![\xi]\!]_{\mathcal{A}_s} = s}} \sum_{r \in \mathrm{runs}_G^\nu(\xi)} \prod_{\substack{\pi \in \mathrm{pos}(\xi) : \\ \mathrm{rule}_r^\pi = A \to f(B_1, \ldots, B_k)}} p(A \to f(B_1, \ldots, B_k)) \qquad \text{Exact}$$

Alternative, feasible parsing objectives are used in practise:

$$P(t, s \mid \mathbb{G}, p) \approx \max_{\substack{\xi \in T_\Sigma : \\ [\![\xi]\!]_{\mathcal{A}_t} = t \\ [\![\xi]\!]_{\mathcal{A}_s} = s}} \max_{r \in \mathrm{runs}_G^\nu(\xi)} \prod_{\substack{\pi \in \mathrm{pos}(\xi) : \\ \mathrm{rule}_r^\pi = A \to f(B_1, \ldots, B_k)}} p(A \to f(B_1, \ldots, B_k)) \qquad \text{Viterbi}$$

Parsing objectives

The probability of a parse tree is obtained by summing over the probabilities of all runs on all derivation trees for this parse tree:

$$P(t, s \mid \mathbb{G}, p) = \sum_{\substack{\xi \in T_\Sigma : \\ [\![\xi]\!]_{\mathcal{A}_t} = t \\ [\![\xi]\!]_{\mathcal{A}_s} = s}} \sum_{r \in \mathrm{runs}_G^\forall(\xi)} \prod_{\substack{\pi \in \mathrm{pos}(\xi) : \\ \mathrm{rule}_r^\pi = A \to f(B_1, \ldots, B_k)}} p(A \to f(B_1, \ldots, B_k)) \qquad \text{Exact}$$

Alternative, feasible parsing objectives are used in practise:

$$P(t, s \mid \mathbb{G}, p) \approx \max_{\substack{\xi \in T_\Sigma : \\ [\![\xi]\!]_{\mathcal{A}_t} = t \\ [\![\xi]\!]_{\mathcal{A}_s} = s}} \max_{r \in \mathrm{runs}_G^\forall(\xi)} \prod_{\substack{\pi \in \mathrm{pos}(\xi) : \\ \mathrm{rule}_r^\pi = A \to f(B_1, \ldots, B_k)}} p(A \to f(B_1, \ldots, B_k)) \qquad \text{Viterbi}$$

$$P(t, s \mid \mathbb{G}, p) \approx \max_{\substack{\xi \in T_\Sigma : \\ [\![\xi]\!]_{\mathcal{A}_t} = t}} \max_{r \in \mathrm{runs}_{G_s^\xi}^\forall(\xi)} \prod_{\substack{\pi \in \mathrm{pos}(\xi) : \\ \mathrm{rule}_r^\pi = A \to f(B_1, \ldots, B_k)}} q(A \to f(B_1, \ldots, B_k))$$

Variational/max-rule-product

E.g. variational:

$$q([A, q_0] \to f([B_1, q_1], \ldots, [B_k, q_k]))$$

$$= \sum_{\substack{A' \to f(B'_1, \ldots, B'_k): \\ A' \to f(B'_1, \ldots, B'_k) \text{ refines } A \to f(B_1, \ldots, B_k)}} \frac{\text{out}([A', q_0]) \cdot p([A', q_0] \to f([B'_1, q_1], \ldots, [B'_k, q_k]}{\text{in}([S, \_])}$$

Statistics on induced base grammars on TiGerHN08

	nonterminals	rules	lexical rules	coverage dev. set
LCFRS_{ho}	767	50,153	28,080	78.3%
$\text{LCFRS}_{\text{r2}\ell}$	817	49,297	28,080	76.5%
$\text{hybrid}_{\text{child}}$	288	39,123	28,080	82.9%
$\text{hybrid}_{\text{strict}}$	32,281	108,957	28,080	50.0%

Based on Gebhardt [2018].

Results TiGerHN08 dev. set for sentences length ≤ 40.

Objective	F1 (disc)	EM	F1-fun	F1 (disc)	EM	F1-fun
	LCFRS head-outward			LCFRS right-to-left		
base-Viterbi	68.29 (22.55)	28.21	41.73	70.36 (23.00)	30.06	43.15
fine-Viterbi	76.59 (29.01)	35.87	63.45	77.32 (30.94)	36.83	65.48
variational	79.09 (33.17)	41.30	67.23	79.04 (34.32)	40.85	68.74
max-rule-prod.	**79.44** (33.74)	**41.73**	67.51	79.21 (**34.54**)	40.95	**68.83**
base-500-rerank	74.09 (29.31)	36.77	55.65	74.52 (28.82)	36.49	56.15
	$\text{hybrid}_{\text{child}}$			$\text{hybrid}_{\text{strict}}$		
base-Viterbi	63.19 (15.04)	23.89	39.22	69.86 (29.34)	29.63	43.24
fine-Viterbi	76.56 (29.66)	39.27	65.03	73.34 (34.47)	33.95	61.06
variational	77.48 (30.53)	40.79	66.96	73.94 (33.75)	35.28	62.34
max-rule-prod.	77.69 (30.45)	41.18	67.05	73.99 (34.02)	35.48	62.37
base-500-rerank	69.30 (25.61)	31.82	52.16	72.53 (32.98)	33.68	55.41

Based on Gebhardt [2018].

More recent experiments with dummy counts

Hybrid grammars Strict markov v1 h1, prior: 1.0, merge 90%, 5 S/M cycles

seed	0	1	2	3	average	median	variance
base-Viterbi			**72.18**				
fine-Viterbi	79.81	79.26	79.48	79.57	79.53	79.53	5.18E-02
variational	80.46	80.29	80.31	80.34	80.35	80.33	5.80E-03
max-rule-prod.	80.59	80.64	80.47	80.76	**80.62**	**80.62**	1.44E-02
base-500-rerank	79.23	79.19	79.23	79.39	79.26	79.23	7.87E-03

Evaluation on the test sets.

	Method	TiGerSPMRL F1 `spmrl/proper`	TiGerHN08 $\ell \leq 40$ F1	EM	F1-fun
Hall and Nivre [2008]	dep2const	- / -	79.93	-	-
Fernández-González and Martins [2015]	dep2const	80.62 / -	**85.53**	-	-
Corro et al. [2017]	dep2const	- / **81.63**	-	-	-
Maier [2015]	SR-swap	- / -	79.52	44.32	-
Maier and Lichte [2016]	SR-swap	- / 76.46	80.02	45.11	-
Coavoux and Crabbé [2017]	SR-gap	**81.50** / **81.60**	85.11	-	-
Stanojević and Garrido Alhama [2017]	SR-adj-swap	- / **81.64**	84.06	-	-
here LCFRS: head-outward/max-rule-product		75.00 / 75.08	79.29	42.55	67.25
here hybrid grammar: child/max-rule-product		72.91 / 72.98	77.68	41.28	66.72
† van Cranenburgh et al. [2016]	DOP	- / -	78.2	40.0	68.1
† **here** LCFRS: head-outward/max-rule-product		- / -	76.91	39.22	64.91
† **here** hybrid grammar: child/max-rule-product		- / -	75.66	38.40	64.66

Based on Gebhardt [2018].

X	Y	reduct	mle	X-LA mle
CFG	sentence	sentence $\cap$ CFG	EM	EM
	cont. phrase structure tree/ proj. dependency tree	read-off	rfe	EM
LCFRS	sentence	sentence $\cap$ LCFRS	EM	EM
	phrase structure tree/ dependency tree	read-off	rfe	EM
LCFRS/sDCP hybrid grammar	sentence	sentence $\cap$ LCFRS	EM	EM
	tree	tree $\cap$ sDCP	EM	EM
	hybrid tree	hybrid tree $\cap$ hybrid grammar	EM	EM
	derivation tree	identity	rfe	EM

Bibliography I

A. Böhmová, J. Hajič, E. Hajičová, and B. Hladká. The Prague dependency treebank. In A. Abeillé, editor, *Treebanks: Building and and Using Parsed Corpora*, chapter 7, pages 103–127. Kluwer Academic Press, 2003. doi: 10.1007/978-94-010-0201-1_7.

S. Brants, S. Dipper, P. Eisenberg, S. Hansen-Schirra, E. König, W. Lezius, C. Rohrer, G. Smith, and H. Uszkoreit. TIGER: Linguistic interpretation of a german corpus. *Res. Lang. Comput.*, 2(4), 2004. doi: 10.1007/s11168-004-7431-3.

M. Coavoux and B. Crabbé. Incremental discontinuous phrase structure parsing with the gap transition. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 1259–1270, Valencia, Spain, April 2017. URL `https://www.aclweb.org/anthology/E17-1118`.

Bibliography II

C. Corro, J. Le Roux, and M. Lacroix. Efficient discontinuous phrase-structure parsing via the generalized maximum spanning arborescence. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1644–1654, Copenhagen, Denmark, September 2017. URL https://www.aclweb.org/anthology/D17-1172.

M.-C. de Marneffe and C. D. Manning. Stanford typed dependencies manual. Technical report, Stanford University, 2008. URL http://www-nlp.stanford.edu/software/dependencies_manual.pdf.

D. Fernández-González and A. F. T. Martins. Parsing as reduction. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1523–1533, Beijing, China, July 2015. URL https://www.aclweb.org/anthology/P15-1147.

M. Forst, N. Bertomeu, B. Crysmann, F. Fouvry, S. Hansen-Schirra, and V. Kordoni. Towards a dependency-based gold standard for german parsers. In *Proceedings of the 5th Workshop on Linguistically Interpreted Corpora*, 2004.

Bibliography III

K. Gebhardt. Generic refinement of expressive grammar formalisms with an application to discontinuous constituent parsing. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 3049–3063, Santa Fe, New Mexico, USA, Aug. 2018. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/C18-1258`.

K. Gebhardt, M.-J. Nederhof, and H. Vogler. Hybrid grammars for parsing of discontinuous phrase structures and non-projective dependency structures. *Computational Linguistics*, 2017. doi: 10.1162/COLI_a_00291. accepted for publication.

J. Hall and J. Nivre. Parsing discontinuous phrase structure with grammatical functions. In B. Nordström and A. Ranta, editors, *Advances in Natural Language Processing*, pages 169–180, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-85287-2. doi: 10.1007/978-3-540-85287-2_17.

W. J. Hutchins and H. L. Somers. *An introduction to machine translation*. London: Academic Press, 1992. ISBN: 0-12-362830-X, `http://www.hutchinsweb.me.uk/IntroMT-TOC.htm`

Bibliography IV

A. K. Joshi and Y. Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–123. Springer-Verlag, 1997.

D. Jurafsky and J. H. Martin. *Speech and Language Processing – An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice-Hall, 2000.

A. Koller and M. Kuhlmann. A generalized view on parsing and translation. In *Proceedings IWPT 2011*, 2011. URL `http://www.aclweb.org/anthology/W/W11/W11-2902.pdf`.

M. Kuhlmann. Mildly non-projective dependency grammar. *Computational Linguistics*, 39(2):355–387, 2013.

M. Kuhlmann and M. Möhl. Mildly context-sensitive dependency languages. In J. A. Carroll, A. van den Bosch, and A. Zaenen, editors, *ACL 2007, Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics, June 23-30, 2007, Prague, Czech Republic*. The Association for Computational Linguistics, 2007. URL `http://aclweb.org/anthology/P07-1021`.

Bibliography V

M. Kuhlmann and G. Satta. Treebank grammar techniques for non-projective dependency parsing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, EACL '09, pages 478–486, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1609067.1609120.

M. Kuhlmann, C. Gómez-Rodríguez, and G. Satta. Dynamic programming algorithms for transition-based dependency parsers. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pages 673–682, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-87-9. URL http://dl.acm.org/citation.cfm?id=2002472.2002558.

Bibliography VI

W. Maier. Discontinuous incremental shift-reduce parsing. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1202–1212, Beijing, China, July 2015. URL https://www.aclweb.org/anthology/P15-1116.

W. Maier and L. Kallmeyer. Discontinuity and non-projectivity: Using mildly context-sensitive formalisms for data-driven parsing. In *Tenth International Workshop on Tree Adjoining Grammar and Related Formalisms*, pages 119–126, 2010.

W. Maier and T. Lichte. Discontinuous parsing with continuous trees. In *Proceedings of the Workshop on Discontinuous Structures in Natural Language Processing*, pages 47–57, San Diego, California, June 2016. URL https://www.aclweb.org/anthology/W16-0906.

W. Maier and A. Søgaard. Tree-banks and mildly context-sensitivity. In *Proc. of Formal Grammar 2008*, pages 61–76, 2008. URL http://web.stanford.edu/group/cslipublications/cslipublications/FG/2008/maier.pdf.

Bibliography VII

M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1994. URL `http://dl.acm.org/citation.cfm?id=972475`.

J. Nivre. Algorithms for deterministic incremental dependency parsing. *Computational Linguistics*, 34(4):513–553, 2008. doi: 10.1162/coli.07-056-R1-07-027.

J. Nivre. Non-projective dependency parsing in expected linear time. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 351–359, Suntec, Singapore, August 2009. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P09-1040`.

W. Skut, B. Krenn, T. Brants, and H. Uszkoreit. An annotation scheme for free word order languages. In *Fifth Conference on Applied Natural Language Processing*, pages 88–95, 1997. doi: 10.3115/974557.974571. Washington, DC, USA, March–April.

Bibliography VIII

G. Smith. A brief introduction to the tiger treebank, version 1. Technical report, Universität Potsdam, 2003. URL http://www.ims.uni-stuttgart.de/forschung/ressourcen/korpora/TIGERCorpus/annotation/tiger_introduction.pdf.

M. Stanojević and R. Garrido Alhama. Neural discontinuous constituency parsing. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 1666–1676, Copenhagen, Denmark, September 2017. URL https://www.aclweb.org/anthology/D17-1174.

A. van Cranenburgh, R. Scha, and R. Bod. Data-oriented parsing with discontinuous constituents and function tags. *Journal of Language Modelling*, 4(1):57–111, 2016. doi: 10.15398/jlm.v4i1.100.