

Tools for parsing natural languages

Bertrand Gaiffe, Guy Perrier

July 1, 2004

Contents

1	Context-free grammars	5
1.1	What is a Context Free grammar ?	5
1.1.1	Putting parentheses around words	5
1.1.2	Definition of context-free grammars	5
1.2	Parsing with CF grammars	6
1.2.1	Results of parsing	6
1.2.2	Parsing as intersection : the overall framework	9
1.2.3	Parsing as deduction	10
1.2.4	CKY parsing	10
1.2.5	What was going on there ?	12
1.2.6	Parsing as deduction	12
1.2.7	The “real” CKY	12
1.2.8	A bad generalization of CKY	13
1.2.9	Generalized CKY	13
1.2.10	Earley algorithm	14
1.2.11	What was going on there ?	16
1.3	Chart Parsing	17
2	Feature structures	19
2.0.1	Feature structures as information structures	19
2.0.2	Unification and subsumption	22
3	Unification grammars	25
3.0.3	Adding equations to a CF grammar	25
3.0.4	Generalisation	25
3.0.5	The intersection grammar	26
3.0.6	Expressiveness	28
3.0.7	Parsing algorithms	28
3.0.8	CKY for unification grammars	30
3.0.9	Earley for unification grammars	30
3.0.10	Offline parsability	30
3.0.11	Extraction of the results	31
4	Parsing with polarities	33
4.1	Introduction	33
4.2	Polarized Unification Grammars	36
4.3	The example of Categorical Grammars	38
4.3.1	AB grammars	39
4.3.2	Translation into the PUG formalism	40
4.4	Expressivity and limitations of PUG	42

5	Underspecification and tree descriptions	45
5.1	Introduction	45
5.2	Polarized Tree Description Grammars	47
5.2.1	The form of polarized tree descriptions	48
5.2.2	The models of tree descriptions	48
5.2.3	Minimal and neutral models of tree descriptions	50
5.2.4	Computation of grammatical models by neutralizing polarities	52
5.3	The example of Lambek Grammars	53
5.3.1	The logical presentation of Lambek Grammars	53
5.3.2	Translation in the PTDG formalism	53
5.4	Possible enrichment of the PTDG formalism	55

Chapter 1

Context-free grammars

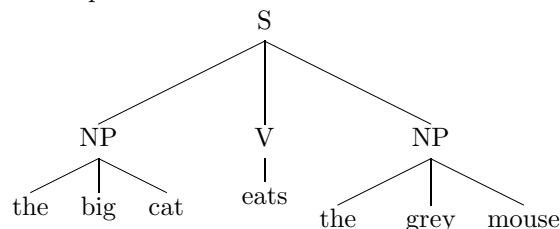
1.1 What is a Context Free grammar ?

1.1.1 Putting parentheses around words

When one faces a sentence in a natural language, what he very probably is interested in is the meaning of the sentence. For instance the sentence : “the big cat eats the grey mouse” means something like *there is a cat, this cat is big, there is a mouse, this mouse is grey, the cat eats the mouse*. Whatever the precise representation of the meaning we choose, what is important is that the cat is big the mouse is grey and it is the cat that eats the mouse. In order to get this semantic representation, we thus somehow parenthesize the utterance :

[the big cat] eats [the grey mouse] so that we make sure to combine “big” with “cat” and “grey” with “mouse”. Moreover, we know that *the big cat* is a noun phrase, *the grey mouse* is also a noun phrase and *the big cat eats the grey mouse* is a sentence.

Another way to represent this parenthetic structure is a tree such as :



This tree exactly says that the whole thing is a sentence, that *the big cat* goes together to form a noun phrase and *the grey mouse* is also a noun phrase.

As we will see, context free grammars are precisely a means to describe such trees.

1.1.2 Definition of context-free grammars

Context free grammars are devices dedicated to generating trees whose leaves are words of a language.

They are defined as $\langle N, T, X, R \rangle$ where :

- N is a (finite and non empty) set of symbols called non-terminals
- T is a (finite and non empty) set of symbols called terminals ($N \cap T = \emptyset$)
- X is the called the axiom ($X \in N$)
- R is a (finite) set of rewriting rules such as : $Y \longrightarrow \alpha$ with Y a non-terminal and α is a list of terminals and non terminals¹. If α is the empty list, we write the rule $Y \longrightarrow \epsilon$.

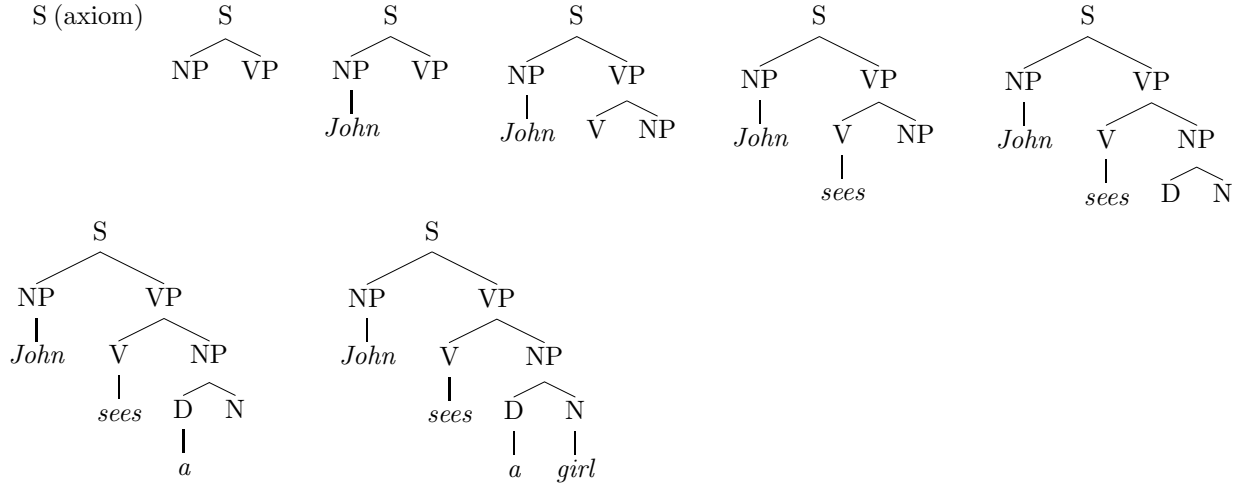
An example grammar is :

¹We write lists of elements of T and NT with greek letters, terminals in lowercase and non terminal in uppercase

- $N = \{S, NP, PP, V, N, D\}$
- $T = \{ \text{John, a, girl, with, sees, telescope} \}$
- axiom = S
- $R = \{ S \rightarrow NP VP, NP \rightarrow \text{John}, NP \rightarrow D N, NP \rightarrow NP PP, VP \rightarrow V NP, VP \rightarrow NP VP PP \}$

The way it works consists in beginning with the axiom and using the rules in R to expand non-terminal until there is no non-terminal to expand anymore.

As an example, we can generate the sentence *John sees a girl* by the following steps :



Context free grammars produce trees. Sometimes however, we are only interested in the yield of these trees.

If this is the case, we can view the preceding derivation as $\boxed{S}, \boxed{NP VP}, \boxed{John VP}, \boxed{John V NP}, \boxed{John sees NP}, \boxed{John sees D N}, \boxed{John sees a N}, \boxed{John sees a girl}.$

We introduce some notations we will need further :

- a non terminal X rewrites in one step as a list of symbols (non terminals or terminals) if there exists a rule in R that says so. We write this as $X \rightarrow \alpha$
- if we apply at least one more rule to an element in α then X rewrites in a new list of symbols in two steps, we write this $X \xrightarrow{2} \beta$
- if X rewrites in a list of symbols in some (finite) number of steps, we then write $X \xrightarrow{*} \gamma$.

1.2 Parsing with CF grammars

In the previous paragraph, we explained how a grammar generates trees whose leaves are words. The yield of the tree is a sentence. The parsing problem consists in finding the trees that the grammar generates for some given sentence at their leaves.

1.2.1 Results of parsing

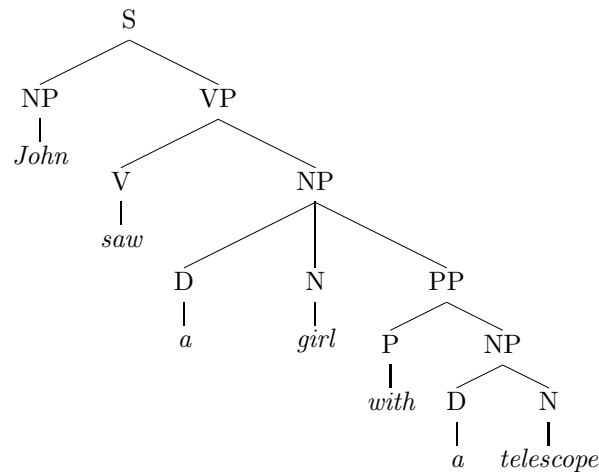
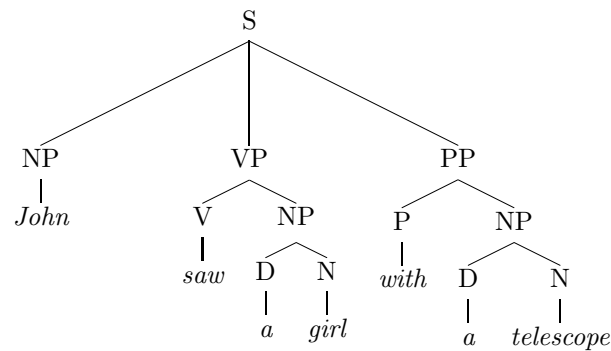
Suppose we analyse the sentence :

John saw a girl with a telescope
with the following grammar :

S	→	NP VP
S	→	NP VP PP
VP	→	V NP
PP	→	P NP
NP	→	D N
NP	→	<i>John</i>
V	→	<i>saw</i>
D	→	<i>a</i>
N	→	<i>girl</i>
N	→	<i>telescope</i>
P	→	<i>with</i>

There are two possible results : one corresponds to the interpretation that *John uses a telescope in order to see the girl*, and the other corresponds to *the girls carries a telescope and John sees her*.

Those two analyses correspond to the following trees :



These two trees are the results we would like to get at the end of the parsing of the sentence. More generally, the result of parsing a sentence is a set of trees (this set may be empty when the sentence cannot be analysed by the grammar).

As we saw in the introduction, Context Free Grammars are precisely dedicated to generating trees. Therefore a good idea would be to describe the results as a context free grammar. Moreover, the grammar that generates the results has to resemble the original grammar. It cannot be too close however, if we wrote our results as :

S	→	NP VP
S	→	NP VP PP
NP	→	<i>John</i>
VP	→	V NP
V	→	<i>saw</i>
NP	→	D N
NP	→	D N PP
D	→	<i>a</i>
N	→	<i>girl</i>
PP	→	P NP
P	→	<i>with</i>
NP	→	D N
D	→	<i>a</i>
N	→	<i>telescope</i>

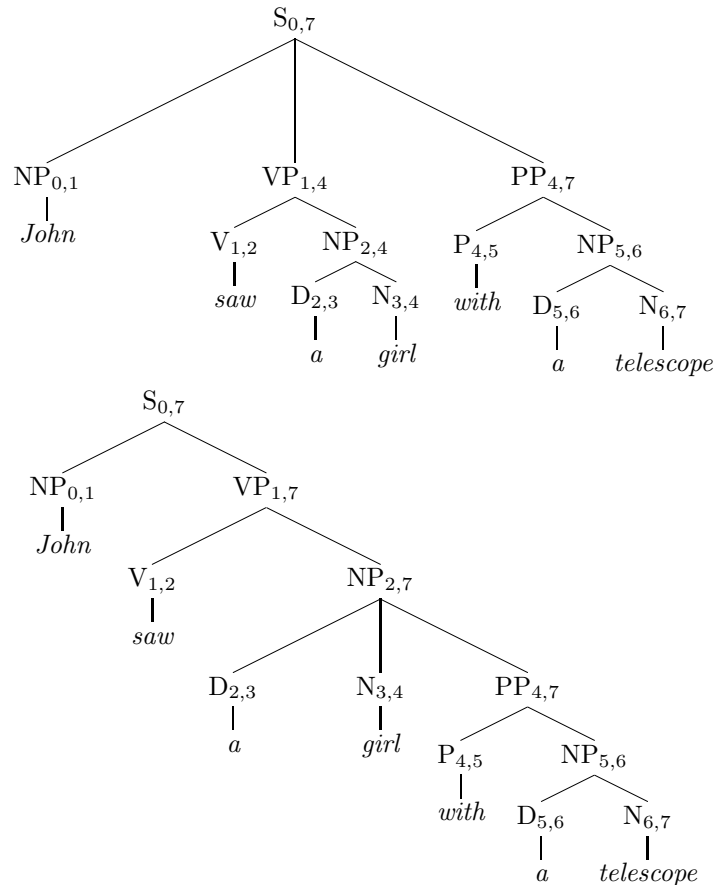
This grammar overgenerates as compared to the two trees we intended as a result. In particular, this grammar generates “the girl with a telescope saw a man”. The reason is that we do not make any difference between the different occurrences of the NPs in the result.

A good way to make them different is to index them relatively to the string to parse. The first NP in both trees corresponds to the first word of the sentence. The last NP (a telescope) correspond to the last two words, etc.

If we index the string to parse as follows :

0 John 1 saw 2 a 3 girl 4 with 5 a 6 telescope 7 .

Then a way to describe our results is :



As now each non-terminal (and each terminal) is decorated with the indexes of the string that it spans, we can easily describe the results by means of a context free grammar :

$S_{0,7}$	$\longrightarrow$	$NP_{0,1} VP_{1,7}$
$S_{0,7}$	$\longrightarrow$	$NP_{0,1} VP_{1,4} PP_{4,7}$
$NP_{0,1}$	$\longrightarrow$	$John_{0,1}$
$VP_{1,4}$	$\longrightarrow$	$V_{1,2} NP_{2,4}$
$V_{1,2}$	$\longrightarrow$	$saw_{1,2}$
$NP_{2,4}$	$\longrightarrow$	$D_{2,3} N_{3,4}$
$PP_{4,7}$	$\longrightarrow$	$P_{4,5} NP_{5,7}$
$D_{2,3}$	$\longrightarrow$	$a_{2,3}$
$N_{3,4}$	$\longrightarrow$	$girl_{3,4}$
$P_{4,5}$	$\longrightarrow$	$with_{4,5}$
$NP_{5,7}$	$\longrightarrow$	$D_{5,6} N_{6,7}$
$D_{5,6}$	$\longrightarrow$	$a_{5,6}$
$N_{6,7}$	$\longrightarrow$	$telescope_{6,7}$
$VP_{1,7}$	$\longrightarrow$	$V_{1,2} NP_{2,7}$
$NP_{2,7}$	$\longrightarrow$	$D_{2,3} N_{3,4} PP_{4,7}$

This grammar only generates our two analyses trees. Moreover it factorizes common parts between the two trees.

1.2.2 Parsing as intersection : the overall framework

In the preceeding section, we chose to present the results of the parsing as a context free grammar. There exists a simple way to obtain such a grammar : the idea is simply to annotate each rule of the initial grammar with indexes in a sensible way.

Suppose a rule in which the right part contains only non-terminals, that is such a rule as :

$$A \longrightarrow B C D \dots K$$

what such rule says is that A rewrites as B C D ... K in that order. Therefore, we can produce annotations of this rules such as

$$A_{i,j} \longrightarrow B_{i,k} C_{k,l} D_{l,m} \dots K_{x,j}$$

what we say there is that B begins at the same index as A does, B finishes where C starts, etc and finally K finishes where A finishes. We do that with all possible indexes (provided that the first index of each non-terminal is smaller or equal to the second). In what concerns the terminals, the rule is a lot more strict : the only allowed indexes are those of the terminal in the input string.

Finally, the axiom of this grammar is the axiom of the original grammar with the indexes 0 and n with n the last index of the input string.

The (big) grammar we obtain describes the intersection of our original CF grammar and of the string to parse. If it ever produces something, this something is exactly the parse trees of section 1.2.1

The problem now is twofold :

- the “Big” Intersection Grammar (henceforth the intersection grammar is only a partial result : it does not directly tell us whether the parsed sentence is actually generated by the grammar.
- in the case that the sentence is actually syntactically correct, providing the user with the full intersection grammar is not sympathetic, we would like a smaller grammar as a result (the problem of the intersection grammar is that it contains a lot of unusefull rules !)

What we know now is that the results of the parsing is a part of the intersection grammar². The algorithms we will look at thus aim at producing grammars rules that are rules of the intersection grammar but not all of them. Moreover, these algorithm have to answer whether the string to parse is actually recognized by the original grammar and as we saw, this question is equivalent to answering whether the intersection grammar yields the input string.

²We can view the intersection grammar as the space into which we are looking for results.

1.2.3 Parsing as deduction

1.2.4 CKY parsing

Testing the intersection grammar

In order to test whether the intersection grammar actually yields something, a simple bottom up technique can be used. The idea is to keep rules that actually yield words with the correct indexes. More formally : a rule such as $A_{i,j} \rightarrow \gamma$ as to be kept only if $\gamma \xrightarrow{*} w_i \cdots w_{j+1}$. Such a test can be done bottom up : if every symbol in the right part of the rule is ok, then the left part is also ok. In what concerns terminals, there are ok provided they appear at the correct place in the input string.

CKY algorithm directly corresponds to this idea.

Recall that we aim at obtaining a CF grammar as a result. We already know that this result is a part of the intersection grammar. Moreover we want an answer to the question whether this grammar produces something.

CKY algorithm [Kas65, You67] aims at producing only a part of the intersection grammar. It proceeds from bottom to top³. The algorithm contains two parts : an initialization phase and a loop.

Initialisation :

For each terminal appearing in the input string, produce a grammar rule such as :

$$w_{i,i+1} \rightarrow w$$

if and only if, w actually appears in the input string between indexes i and $i + 1$.

Loop :

So long as we can produce new grammar rules, do

produce a grammar rule such as :

$$X_{i,j} \rightarrow A_{i,k} B_{k,l} \cdots K_{m,j}$$

if and only if at least a rule whose left is $A_{i,l}$ and at least a rule whose left is $B_{k,l} \dots$ a rule whose left is $K_{m,j}$ have already been produced, and a rule $X \rightarrow A B \cdots K$ exists in the grammar.

Result : If the preceding loop produced at least a rule whose left side is $S_{0,n}$ then claim that you got an analysis of the sentence and that the parse trees are generated by the resulting grammar with axiom $S_{0,n}$.

Lets try that on our favorite example :

0 John 1 saw 2 a 3 girl 4 with 5 a 6 telescope 7 .
we first produce the rules for the terminals :

$John_{0,1}$	$\rightarrow$	$John$
$saw_{1,2}$	$\rightarrow$	saw
$a_{2,3}$	$\rightarrow$	a
$girl_{3,4}$	$\rightarrow$	$girl$
$with_{4,5}$	$\rightarrow$	$with$
$a_{5,6}$	$\rightarrow$	a
$telescope_{6,7}$	$\rightarrow$	$telescope$

and then we begin the loop :
we build :

$$NP_{0,1} \rightarrow John_{0,1}$$

because a rule whose left is $John_{0,1}$ already exists and the grammar contains $NP \rightarrow John$.
for the same reason, we build :

$$NP_{2,4} \rightarrow a_{2,3} girl_{3,4}$$

and

$$NP_{5,7} \rightarrow a_{5,6} telescope_{6,7}$$

we proceed methodically and finally get :

³This is not the original presentation of the algorithm. We will see the original presentation latter

$\text{John}_{0,1}$	$\longrightarrow$	John
$\text{saw}_{1,2}$	$\longrightarrow$	saw
$\text{a}_{2,3}$	$\longrightarrow$	a
$\text{girl}_{3,4}$	$\longrightarrow$	girl
$\text{with}_{4,5}$	$\longrightarrow$	with
$\text{a}_{5,6}$	$\longrightarrow$	a
$\text{telescope}_{6,7}$	$\longrightarrow$	telescope

$\text{NP}_{0,1}$	$\longrightarrow$	$\text{John}_{0,1}$
$\text{V}_{1,2}$	$\longrightarrow$	$\text{saw}_{1,2}$
$\text{D}_{2,3}$	$\longrightarrow$	$\text{a}_{2,3}$
$\text{N}_{3,4}$	$\longrightarrow$	$\text{girl}_{3,4}$
$\text{P}_{4,5}$	$\longrightarrow$	$\text{with}_{4,5}$
$\text{D}_{5,6}$	$\longrightarrow$	$\text{a}_{5,6}$
$\text{N}_{6,7}$	$\longrightarrow$	$\text{telescope}_{6,7}$

$\text{NP}_{2,4}$	$\longrightarrow$	$\text{D}_{2,3} \text{ N}_{3,4}$
$\text{VP}_{1,4}$	$\longrightarrow$	$\text{V}_{1,2} \text{ NP}_{2,4}$
$\text{S}_{0,4}$	$\longrightarrow$	$\text{NP}_{0,1} \text{ VP}_{1,4}$
$\text{NP}_{5,7}$	$\longrightarrow$	$\text{D}_{5,6} \text{ N}_{6,7}$
$\text{PP}_{4,7}$	$\longrightarrow$	$\text{P}_{4,5} \text{ NP}_{5,7}$

$\text{VP}_{1,7}$	$\longrightarrow$	$\text{V}_{1,2} \text{ NP}_{2,4} \text{ PP}_{4,7}$
$\text{NP}_{2,7}$	$\longrightarrow$	$\text{NP}_{2,4} \text{ PP}_{4,7}$

$\text{VP}_{1,7}$	$\longrightarrow$	$\text{V}_{1,2} \text{ NP}_{2,7}$
-------------------	-------------------	-----------------------------------

$\text{S}_{0,7}$	$\longrightarrow$	$\text{NP}_{0,1} \text{ VP}_{1,7}$
------------------	-------------------	------------------------------------

Now, we have got a rule with left $\text{S}_{0,7}$ so the sentence was successfully parsed.

Why is this algorithm correct ?

In order to prove that this algorithm is correct, we have to prove that :

1. it does not build result trees that are incorrect.
 - this cannot happen because the rules we build are rules of the intersection grammar.
2. the algorithm does not loop infinitely :
 - no risk : there is a finite number of rules in the intersection grammar.
3. when the algorithm pretends there is a result, it is actually the case ;
4. all possible analyses are actually produced.

We have first to prove that when the algorithm says that there is a result, it is actually the case. In order to prove that, we are going to prove that when we have generated a rule whose left part is $X_{i,j}$ then $X_{i,j} \xrightarrow{*} w_i \cdots w_j$

We are going to prove it by recurrence on the height of the analyses trees.

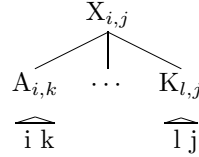
This is obviously the case for the rules we generate for the terminals which correspond to trees of height 1.

Suppose now that it is the case for derivations of height up to n . We are going to prove that it is the case for derivations of height $n + 1$.

If a rule produces a tree of height $n + 1$ then all the terminals and non terminal in its right part produce trees of height smaller or equal to n . Therefore each of them verifies the induction hypothesis.

As the rule is such as $X_{i,j} \rightarrow A_{i,k} \dots K_{l,j}$ and there exists a rule in the grammar such as $X \rightarrow A \dots K$ then $X_{i,j} \xrightarrow{*} w_i \dots w_j$

Schematically :



The same type of demonstration shows that we actually get all the possible solutions

1.2.5 What was going on there ?

In the algorithm we gave, we did not really rely on the entire form of the rules we produced. The only important thing was that there already existed at least a produced rule whose left part was of the form $X_{i,j}$.

Somehow all produced rules with the same left part were considered equivalent in the algorithm. As we just showed, all the rules with left = $X_{i,j}$ correspond to a demonstration that $X \xrightarrow{*} w_{i+1} \dots w_j$.

1.2.6 Parsing as deduction

The observation we just made drives us directly to “parsing as deduction”⁴. Under this perspective, we do not directly build the resulting grammar, we just attempt at doing a demonstration that the string to parse is the yield of at least a tree generated by the grammar.

In order to do so, we will rely on so called items. An item of the form $\langle i, X, j \rangle$ stands for a demonstration that $X \xrightarrow{*} w_{i+1} \dots w_j$.

We will also rely on inference rules, for instance :

$$\frac{\langle i, A, j \rangle, \langle j, B, k \rangle}{\langle i, X, k \rangle}$$

if there is a grammar rule $X \rightarrow A B$.

such a rule is sound for the following reason : provided we already proved that $A \xrightarrow{*} w_{i+1} \dots w_j$ and that we also already proved that $B \xrightarrow{*} w_{j+1} \dots w_k$, provided moreover that there is a grammar rule $X \rightarrow A B$, then it is true that $X \xrightarrow{*} w_{i+1} \dots w_k$.

1.2.7 The “real” CKY

The classical presentation of CKY presupposes that the grammar is in Chomsky Normal Form. This form authorizes only two kind of rules :

- $W \rightarrow w$ with W a non terminal and w a terminal
- $Y \rightarrow A B$ with A, B and Y non-terminals.

we will be a little bit more liberal and authorize also rules of the form $A \rightarrow \epsilon$ and $Y \rightarrow A$.

The inference rules are then the following :

$$\frac{}{\langle i, w, i+1 \rangle}$$

if w in the string between i and $i+1$.

$$\frac{}{\langle i, \epsilon, i \rangle}$$

for all values of i in $[0..n]$

$$\frac{}{\langle i, X, j \rangle}$$

$$\frac{}{\langle i, Y, j \rangle}$$

if there is a rule $Y \rightarrow X$ in the grammar

⁴A very good presentation of this approach is in [SSP93]

$$\frac{\langle i, A, j \rangle \quad \langle j, B, k \rangle}{\langle i, Y, k \rangle}$$

if there is a rule $Y \rightarrow A B$ in the grammar.

Provided we apply fairly the rules (that is no applicable rule remains unapplied), this algorithm works for the same reason our building grammar algorithm worked. Moreover, this algorithm can be extended in order to produce the result grammar : we just have to keep for each item the inference rule and the arguments items that produced it, in the particular case of CKY, if we just keep the argument items, we can read the result directly as a grammar.

1.2.8 A bad generalization of CKY

The previous algorithm supposes that grammar rules are at most binary. A first possible idea is to generalize the rule so that it takes into account rules of any length. This gives us the following rules :

- init : $\frac{}{\langle i, w, i+1 \rangle}$ if w appears in the input string between i and $i+1$.
- reduce : $\frac{\langle i, A, j \rangle \langle j, B, k \rangle \dots \langle l, K, j \rangle}{\langle i, X, j \rangle}$ if there is a grammar rule $X \rightarrow A B \dots K$

If we finally get an item of the form $\langle 0, S, n \rangle$ then we can claim that the input string is recognized by the grammar. If we also want the results, all we have to do is to keep track of the different means through which we obtained each item (that is keep track that $\langle i, X, j \rangle$ was obtained from $\langle i, A, j \rangle \langle j, B, k \rangle \dots \langle l, K, j \rangle$) this “keeping track” mechanism produces exactly the result grammar.

An illustration is the following chart we obtain :

7	S	VP	NP		PP	NP	telescope N
6						a D	
5					with P		
4	S	VP	NP	girl N			
3			a D				
2		saw V					
1	John NP						
	0	1	2	3	4	5	6

This generalisation produces correct results, however it is not efficient. In the worst possible case, the completion rule will always be applied for the longest grammar rule, and it will be used as much as possible ; if we count all those possibilities, we find n^{l+1} of them with l the length of the longest grammar rule and n the number of words in the sentence to parse⁵.

1.2.9 Generalized CKY

A better solution is thus to binarize the rules during the parsing itself. The idea is simply that such a rule as :

$$X \rightarrow A B C \dots I J K \text{ is equivalent to : } \begin{cases} X \rightarrow A_B_C \dots _I_J K \\ A_B_C \dots _I_J \rightarrow A_B_C \dots _I J \\ \dots \\ A_B \rightarrow A B \end{cases}$$

⁵This also tells us that the cost of normal CKY grows like n^3

with $A_B, A_B_C, \dots A_B_C \dots L_J$ new non terminals. This is in practice simulated with a dot ($\bullet$) in the rule.

The items are now of the form $\langle i, A \rightarrow \beta \bullet \gamma, j \rangle$ with $A \rightarrow \beta \gamma$ a rule of the grammar. The rules are now :

- reduce :
$$\frac{\langle i, A \rightarrow \alpha \bullet B \beta, j \rangle \langle j, B \rightarrow \gamma \bullet, k \rangle}{\langle i, A \rightarrow \alpha B \bullet \beta, k \rangle}$$
- scan :
$$\frac{\langle i, A \rightarrow \alpha \bullet w \beta, j \rangle}{\langle i, A \rightarrow \alpha w \bullet \beta, j+1 \rangle}$$
 if w is in the input string between j and $j+1$.
- init1 :
$$\frac{}{\langle i, A \rightarrow \epsilon \bullet, i \rangle}$$
 for all rules $A \rightarrow \epsilon$ of the grammar and all i between 0 and n .
- init2 :
$$\frac{}{\langle i, A \rightarrow w \bullet \beta, i+1 \rangle}$$
 for all rules $A \rightarrow w \beta$ of the grammar and all i such that w in the input string between i and $i+1$.

In order to get the results of this parsing algorithm, we have to remember the implicit binarisation of the rules : it is only when the dot is at the end of the rule that we face a real rule of our initial grammar, the other items just correspond to giving indexes to the non terminal that the dot goes through.

1.2.10 Earley algorithm

If we consider again the intersection grammar, Earley's algorithm [Ear70] consists in building the rules in a top down fashion.

The idea is the following : As it is stupid to annotate the axiom with a first index different from 0, we begin with that. Suppose then we have a rule such as $S \rightarrow A B C$, a good idea is to try to recognize A , with first index equal to 0, when A will be recognized, we will know where it finishes, we will then begin to recognize B from that index, and so on until the whole rule is done.

More generally, we will try to recognize a rule $X \rightarrow Y W Z$ from some index i ; at some point in the process we will have treated partly the right hand side (suppose we did Y) and there will be something to be done. We will therefore be in such a situation as : $X_{i,?} \rightarrow Y_{i,k} W_{k,?} Z_{?,?}$ where $?$ means unknown index. In order to make clear what the treated part of the rule is, we will insert a dot in the rule. Our rule is then written : $X_{i,?} \rightarrow Y_{i,k} \bullet_k W_{k,?} Z_{?,?}$

As usual, the algorithm consists in an initialisation and a loop.

initialisation

For each rule of the grammar whose left is the axiom ($S \rightarrow \alpha$), produce a rule $S_{0,?} \rightarrow \bullet_0 \alpha_{0,?}$. Where $\alpha_{0,?}$ stands for a labeling of the first symbol of α with 0 as its first index.

loop

Until no new rule can be produced do :

- if there exists already a rule of the form : $X_{i,?} \rightarrow \alpha \bullet_j w \beta$ with w a terminal. If w appears in the input string between j and $j+1$ then produce the rule $X_{i,?} \rightarrow \alpha w_{j,j+1} \bullet_{j+1} \beta$.
- if there exists already a rule of the form : $X_{i,?} \rightarrow \alpha \bullet_j B \beta$ with B a non-terminal, then produce all the rule $B_{j,?} \rightarrow \bullet_j \beta_{j,?}$ corresponding to grammar rules that define B in the grammar.
- if there already exists two rules such as : $X_{i,?} \rightarrow \alpha \bullet_j B \beta$ and $B_{j,?} \rightarrow \beta_{j,k} \bullet_k$ [the dot is at the end of the rule] then produce the rule $X_{i,?} \rightarrow \alpha B_{j,k} \bullet_k \beta$

If at the end of the loop a rule such as $S_{0,?} \rightarrow \alpha \bullet_n$ exists then claim that the input string is analyzed.

In order to get the result grammar, throw away any rule that has not the dot at its end, then for each remaining rule (the dot is thus at the end) copy the index of the dot as second index of the left part and erase the bullet.

our favorite example :

Initialisation :

Loop :

$$S_{0,?} \longrightarrow \bullet_0 \text{ NP VP}$$

$$\begin{array}{lcl} \text{NP}_{0,?} & \longrightarrow & \bullet_0 \text{ D N} \\ \text{NP}_{0,?} & \longrightarrow & \bullet_0 \text{ D N PP} \\ \text{NP}_{0,?} & \longrightarrow & \bullet_0 \text{ John} \end{array}$$

$$\text{NP}_{0,?} \longrightarrow \text{John}_{0,1} \bullet_1$$

$$S_{0,?} \longrightarrow \text{NP}_{0,1} \bullet_1 \text{ VP}$$

$$\begin{array}{lcl} \text{VP}_{1,?} & \longrightarrow & \bullet_1 \text{ V NP} \\ \text{VP}_{1,?} & \longrightarrow & \bullet_1 \text{ V NP PP} \end{array}$$

$$\text{V}_{1,?} \longrightarrow \bullet_1 \text{ saw}$$

$$\text{V}_{1,?} \longrightarrow \text{saw}_{1,2} \bullet_2$$

$$\begin{array}{lcl} \text{VP}_{1,?} & \longrightarrow & \text{V}_{1,2} \bullet_2 \text{ NP} \\ \text{VP}_{1,?} & \longrightarrow & \text{V}_{1,2} \bullet_2 \text{ NP PP} \end{array}$$

$$\begin{array}{lcl} \text{NP}_{2,?} & \longrightarrow & \bullet_2 \text{ John} \\ \text{NP}_{2,?} & \longrightarrow & \bullet_2 \text{ D N} \\ \text{NP}_{2,?} & \longrightarrow & \bullet_2 \text{ D N PP} \end{array}$$

$$\text{D}_{2,?} \longrightarrow \bullet_2 \text{ a}$$

$$\text{D}_{2,?} \longrightarrow \text{a}_{2,3} \bullet_3$$

$$\begin{array}{lcl} \text{NP}_{2,?} & \longrightarrow & \text{D}_{2,3} \bullet_3 \text{ N} \\ \text{NP}_{2,?} & \longrightarrow & \text{D}_{2,3} \bullet_3 \text{ N PP} \end{array}$$

$$\begin{array}{lcl} \text{N}_{3,?} & \longrightarrow & \bullet_3 \text{ telescope} \\ \text{N}_{3,?} & \longrightarrow & \bullet_3 \text{ girl} \end{array}$$

$$\text{N}_{3,?} \longrightarrow \text{girl}_{3,4} \bullet_4$$

$$\begin{array}{lcl} \text{NP}_{2,?} & \longrightarrow & \text{D}_{2,3} \text{ N}_{3,4} \bullet_4 \\ \text{NP}_{2,?} & \longrightarrow & \text{D}_{2,3} \text{ N}_{3,4} \bullet_4 \text{ PP} \end{array}$$

$$\text{VP}_{1,?} \longrightarrow \text{V}_{1,2} \text{ NP}_{2,4} \bullet_4$$

$$S_{0,?} \longrightarrow \text{NP}_{0,1} \text{ VP}_{1,4} \bullet_4$$

$$\text{VP}_{1,?} \longrightarrow \text{V}_{1,2} \text{ NP}_{2,4} \bullet_4 \text{ PP}$$

PP _{4,?}	→	• ₄ P NP
P _{4,?}	→	• ₄ <i>with</i>
P _{4,?}	→	<i>with</i> _{4,5} • ₅
PP _{4,?}	→	P _{4,5} • ₅ NP
NP _{5,?}	→	• ₅ <i>John</i>
NP _{5,?}	→	• ₅ D N
D _{5,?}	→	• ₅ <i>a</i>
D _{5,?}	→	<i>a</i> _{5,6} • ₆
NP _{5,6}	→	D _{5,6} • ₆ N
N _{6,?}	→	• ₆ <i>telescope</i>
N _{6,?}	→	<i>telescope</i> _{6,7} • ₇
NP _{5,?}	→	D _{5,6} N _{6,7} • ₇
PP _{4,?}	→	P _{4,5} NP _{5,7} • ₇

VP _{1,?}	→	V _{1,2} NP _{2,4} PP _{4,7} • ₇
NP _{2,?}	→	D _{2,3} N _{3,4} PP _{4,7} • ₇
VP _{1,?}	→	V _{1,2} NP _{2,7} • ₇

S _{0,?}	→	NP _{0,1} VP _{1,7} • ₇
------------------	---	--

We can now claim that the sentence is analyzed, and give as a result the following grammar :

NP _{0,1}	→	<i>John</i> _{0,1}
V _{1,2}	→	<i>saw</i> _{1,2}
D _{2,3}	→	<i>a</i> _{2,3}
N _{3,4}	→	<i>girl</i> _{3,4}
NP _{2,4}	→	D _{2,3} N _{3,4}
VP _{1,4}	→	V _{1,2} NP _{2,4}
S _{0,4}	→	NP _{0,1} VP _{1,4}
P _{4,5}	→	<i>with</i> _{4,5}
D _{5,6}	→	<i>a</i> _{5,6}
N _{6,7}	→	<i>telescope</i> _{6,7}
NP _{5,7}	→	D _{5,6} N _{6,7}
PP _{4,7}	→	P _{4,5} NP _{5,7}
VP _{1,7}	→	V _{1,2} NP _{2,4} PP _{4,7}
NP _{2,7}	→	D _{2,3} N _{3,4} PP _{4,7}
VP _{1,7}	→	V _{1,2} NP _{2,7}
S _{0,7}	→	NP _{0,1} VP _{1,7}

This grammar yields our two results and contains (only) two useless rules.

1.2.11 What was going on there ?

In the preceding algorithm, the rules we build are grouped considering the following informations :

- the rule in the analysis grammar,
- the position of the dot in the rule,
- the index where the recognition of the rule began,
- the index of the dot.

Therefore, what is really important for proving that the grammar accepts the input string is items of the form : $\langle i, X \rightarrow \alpha \bullet \beta, j \rangle$.

We can therefore present Earley's algorithm in its more traditional version, through inference rules :
initialisation

$$\begin{array}{l}
< 0, S \rightarrow \bullet \gamma, 0 > \\
\text{for all rules defining the axiom.} \\
\textbf{loop} \\
< i, X \rightarrow \alpha \bullet w \beta, j > \\
\hline
< i, X \rightarrow \alpha w \bullet \beta, j + 1 > \\
\text{if } w \text{ appears in the input string between } j \text{ and } j + 1 \\
< i, X \rightarrow \alpha \bullet B \beta, j > \\
\hline
< j, B \rightarrow \bullet \delta, j > \\
\text{for all rules that define } B. \\
< i, X \rightarrow \alpha \bullet B \beta, j > < j, B \rightarrow \delta \bullet, k > \\
\hline
< i, X \rightarrow \alpha B \bullet \beta, k >
\end{array}$$

success condition

We are successful in parsing the sentence if we produce an item such as $< 0, S \rightarrow \gamma \bullet, n >$ with n the length of the input string.

Note that the rules of the result grammar are built progressively when applying the reduction rule, there is a general condition and a specific to the end of rule one :

- when the dot advances, we collect the indexes of the terminal or non terminal the dot goes through.
- if the dot arrives at the end of the rule, when we know the indexes of the left non-terminal of the rule (they are the indexes of the item)

This algorithm actually builds a proof that the grammar derives the input sentence. What happens is that an item $< i, X \rightarrow \alpha \bullet \beta, j >$ is a proof that $X \xrightarrow{*} w_{i+1} \cdots w_j \beta$.

1.3 Chart Parsing

We saw principles underlying CKY and Earley's algorithms. However, we still do not have **real** algorithms. We need a mechanism for applying the inference rules on the items. We also need to store the items so that to reuse them when needed.

The chart is a two dimensional array (we actually only use half of it because the first index of each item is equal or less than the second one). In each entry of the array, a set of items is stored. Each inference rule produces new items. These items have to be tested against items in the chart. The structure in which items to be treated is kept is called the agenda. The general algorithm for chart parsing then runs as follow :

1. initialize the agenda with the initial items (initialisation step of the algorithm)
2. repeat until the agenda is empty
 - select an item from the agenda and remove that item from it
 - test applicable rules on this item (together with items of the chart if needed)
 - for each resulting item, if it is not yet in the chart, then add it to the chart and the agenda
3. if a successful item is in that chart, then claim that the analysis is successful.

In order to get results, we also need for each item in the chart, to store the way the item was produced, that is the inference rule applied and the items on which it was applied. From that information, we can build the resulting grammar.

Chapter 2

Feature structures

Our example grammar so far, was really a toy. Suppose we simply add pronouns to it so that we can accept sentences such as :

he saw her

she saw him

The problem is that neither *him saw she* or *her saw he* is an acceptable sentence. We could write our new toy grammar using such non terminals as OBJECT_PRONOUN or SUBJECT_PRONOUN, but what looks like a simpler solution would be to have a PRONOUN non-terminal together with a qualification as “subject” or “object”.

Feature structures are means to represent such things as “object pronoun” or “subject pronoun”.

2.0.1 Feature structures as information structures

As we just mentioned in the introduction, we aim at representing non-terminals to which we could apply functions in order to get values. For instance we would query our pronoun for its syntactic function and get either “subject” or “object” as a result.

The usual notation for such informational structures is :

$$\left[\begin{array}{ll} cat & = NP \\ syntacticFunction & = subject \\ word & = he \\ person & = third \end{array} \right]$$

On such a structure, we can apply functions, for instance,

$$person\left(\left[\begin{array}{ll} cat & = NP \\ syntacticFunction & = subject \\ word & = he \\ person & = third \end{array} \right] \right) = third$$

It is clear then that the order in which we write the attributes and their values makes no difference. Typically,

$$person\left(\left[\begin{array}{ll} cat & = NP \\ syntacticFunction & = subject \\ word & = he \\ person & = third \end{array} \right] \right) = third$$

$$\text{and, } person \left(\begin{array}{lcl} cat & = & NP \\ person & = & third \\ word & = & he \\ syntacticFunction & = & subject \end{array} \right) = third \text{ as well.}$$

Sometimes, we want a deeper structure, instead of having such a flat representation, for instance, we could decide that the syntactic function, the person and the number are a matter of agreement.

We could therefore consider such a feature structure as :

$$\left[\begin{array}{lcl} cat & = & NP \\ word & = & he \\ agreement & = & \left[\begin{array}{lcl} syntacticFunction & = & subject \\ person & = & third \\ number & = & singular \end{array} \right] \end{array} \right]$$

On such nested structures, we can apply successive functions. For instance,

$$person(agreement \left(\begin{array}{lcl} cat & = & NP \\ word & = & he \\ agreement & = & \left[\begin{array}{lcl} syntacticFunction & = & subject \\ person & = & third \\ number & = & singular \end{array} \right] \end{array} \right))) =$$

$$person \left(\begin{array}{lcl} syntacticFunction & = & subject \\ person & = & third \\ number & = & singular \end{array} \right) = third$$

Of course, there may be occasions in which we would like two different means to access the same information. For instance, a feature structure such as :

$$\left[\begin{array}{lcl} cat & = & NP \\ word & = & he \\ agreement & = & \left[\begin{array}{lcl} syntacticFunction & = & subject \\ person & = & third \\ number & = & singular \end{array} \right] \\ number & = & singular \end{array} \right]$$

would perfectly make sense. On such a feature structure, two different paths (namely $\langle agreement\ number \rangle$ and $\langle number \rangle$) access to the same information (namely “singular”).

Adding information into feature structures

So far, we only spoke about accessing information stored in feature structures. A quite natural question is then, how do we add information into a feature structure ? A first consideration is that adding information into a feature structure may lead to non sense.

Typically trying to add “gender feminine” to $\left[\begin{array}{lcl} cat & = & Pronoun \\ gender & = & masculine \\ word & = & he \end{array} \right]$ should not have any meaning.

In the last example of last section we considered :

$$\left[\begin{array}{lcl} cat & = & NP \\ word & = & he \\ agreement & = & \left[\begin{array}{lcl} syntacticFunction & = & subject \\ person & = & third \\ number & = & singular \end{array} \right] \\ number & = & singular \end{array} \right]$$

Suppose now that we only have part of it, namely :

$$\left[\begin{array}{lcl} cat & = & NP \\ word & = & he \\ aggrement & = & \left[\begin{array}{lcl} syntacticFunction & = & subject \\ person & = & third \end{array} \right] \end{array} \right]$$

and suppose moreover that we want to add $\langle aggrement\ number \rangle = singular$, it would be nice that we get our original structure. In order to do that, we need something saying in words : “whatever value $\langle aggrement\ number \rangle$ is, we want this value to be also $\langle number \rangle$ ”.

Such a mecanism exists in feature structure, it is writen with $\square_n$. We then have :

$$\left[\begin{array}{lcl} cat & = & NP \\ word & = & he \\ aggrement & = & \left[\begin{array}{lcl} syntacticFunction & = & subject \\ person & = & third \\ number & = & \square_1 \end{array} \right] \\ number & = & \square_1 \end{array} \right]$$

and if we add $\langle aggrement\ number \rangle = singular$ or if we add $\langle number \rangle = singular$ we will get :

$$\left[\begin{array}{lcl} cat & = & NP \\ word & = & he \\ aggrement & = & \left[\begin{array}{lcl} syntacticFunction & = & subject \\ person & = & third \\ number & = & \square_1 singular \end{array} \right] \\ number & = & \square_1 \end{array} \right]$$

Note that $\square_1$ acts just like a pointer, it corresponds to some unique place into the feature structure : if we go through $\langle aggrement\ number \rangle$ we arrive at the same place as if we go through $\langle number \rangle$. Therefore, the value we see through $\langle number \rangle$ is (also) “singular”.

A path notation

An alternative notation of feature structures relies on the notion of paths. Under this perspective, we can view a feature structure as a set of equations of two possible forms :

- either $path = path$. For instance $\langle aggrement\ gender \rangle = \langle gender \rangle$
- or $path = value$. For instance $\langle aggrement\ gender \rangle = masculine$.

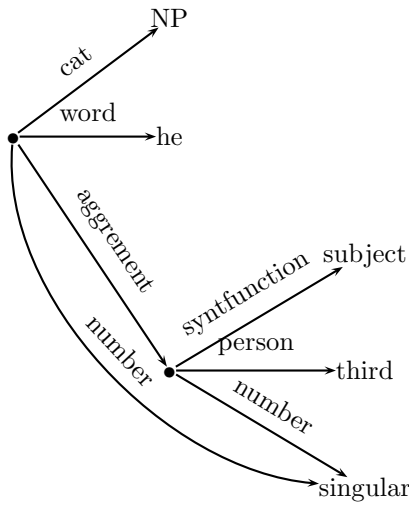
The set of equations is to be understood as a conjunction (a logical “and”) of such equations.

Adding information in a feature structure is then just a matter of adding equations to a set of equations¹.

The graph view of feature structure

We just mentionned a notion of path that may lead to the same point. The natural view of feature structures is thus a (directed) labeled graph. Our feature structure under this view is :

¹An alternative logics for describing feature structures is presented in [Bla00]



This is exactly the notion of model in logics². The bracket notation as well as the path notation are syntax and direct acyclic graphs correspond to the notion of model.

2.0.2 Unification and subsumption

We lack two important notions regarding feature structures. The first one is the notion of subsumption. The second one is the notion of unification.

subsumption

The notion of subsumption corresponds to the idea of containing more or less information. For instance : $[\textit{gender} = \textit{masculine}]$ is more general than $\left[\begin{array}{l} \textit{gender} = \textit{masculine} \\ \textit{number} = \textit{singular} \end{array} \right]$.

This has to do with a notion of implication : $\left[\begin{array}{l} \textit{gender} = \textit{masculine} \\ \textit{number} = \textit{singular} \end{array} \right] \Rightarrow [\textit{gender} = \textit{masculine}]$.

Of course, there exists feature structure that cannot be compared by subsumption, either because the two feature structures contradict each other, or because they contain entirely different informations. The first case is illustrated by : $[\textit{gender} = \textit{masculine}]$ and $[\textit{gender} = \textit{feminine}]$ and the second case is illustrated by $[\textit{gender} = \textit{masculine}]$ and $[\textit{number} = \textit{plural}]$.

Subsumption has the following properties :

- it is reflexive. (A is more general than itself)
- it is anti-symmetric (if A is more general than B and B is more general than A then A is equal to B).
- it is transitive. (if A is more general than B and B is more general than C, then A is more general than C).

it is therefore a partial order.

So far, we only looked at subsumption in intuitive terms, and we did not mention shared paths. If we consider the following feature structures :

$$\left[\begin{array}{l} \textit{aggrement1} = \square_1 [\textit{gender} = \textit{masculine}] \\ \textit{aggrement2} = \square_1 \end{array} \right] \text{ and } \left[\begin{array}{l} \textit{aggrement1} = [\textit{gender} = \textit{masculine}] \\ \textit{aggrement2} = [\textit{gender} = \textit{masculine}] \end{array} \right]$$

We want the second one to be more general than the first because the first contains more information (namely $\langle \textit{aggrement1} \rangle = \langle \textit{aggrement2} \rangle$). This is important because if we add to both feature structures $\langle \textit{aggrement1 number} \rangle = \textit{singular}$ the results will be different.

In order to define formally subsumption, we need the notion of domain of a feature structure. The domain of a feature structure is just the set of features that have a value in the feature structure. Subsumption is then defined recursively as follows :

A subsumes B ($A \sqsubset B$) if and only if :

²this logic is described in [KR86]

1. $l(A) \sqsubset l(B)$ for all l in $\text{domain}(B)$.
2. for each path equation $p = q$ with p and q two paths in A , $p = q$ is also a path equation in B .

Unification

Unifying two feature structures consists in accumulating the information contained in both. Under the logical perspective (feature structures as conjunction of equations), this amounts simply to taking the logical “and” of both formulas. What may happen however is that the resulting formula is inconsistent.

Another view of unification relies on subsumption : the unification of A and B is a feature structure that is more specific than A and more specific than B and that do not contain unnecessary information (technically, this means that it is the more general feature structure that is more specific than A and more specific than B).

The third possible view of unification corresponds to the graph view of feature structures. The algorithm proposed below relies on this graph vision.

The idea is to build a graph that corresponds to both arguments, that is a graph that encodes all information contained in both arguments. The nodes of the resulting graph are labelled by sets of states of the original graphs, for simplicity, we consider states with both a name and a list of nodes associated to that name. The unification algorithm is then :

```

unify(s<->E)
for each e in E do
  if e is final and associated to a constant k then
    if s is final and associated to l different of k then FAIL
    else if s allreadu has a transition (s is not final) then FAIL
    else mark s as final and associate k to it.
  else
    for each transition (a, ea) of e do
      if s has a transition (a,sa<->EA) then if ea not in EA then
        unify(sa<->EA U ea)
      else
        unify(NEW_STATE<->{ea})
        add transition a, NEW_STATE to s
      endif
    end for
  endif
endif
endfor

```

This algorithm has to be called with “unify(NEW_STATE $\leftrightarrow$ {e0, t0})” with e0 and t0 the initial states of both feature structures³.

³More realistic algorithms are described in particular in [MCC00]

Chapter 3

Unification grammars

3.0.3 Adding equations to a CF grammar

A natural way to take feature structures into account is simply to add equations to a context free skeleton. For instance, a fragment of such a grammar could look like :

$NP \longrightarrow Det\ Noun$

Plus an equation such as : $\langle NP\ number \rangle = \langle Det\ number \rangle \langle Noun\ number \rangle$

We would also in such a case have such rules as :

$Det \longrightarrow the$

$\left\{ \begin{array}{l} Det \longrightarrow a \\ \langle Det\ number \rangle = singular \end{array} \right.$

$\left\{ \begin{array}{l} Det \longrightarrow \epsilon \\ \langle Det\ number \rangle = plural \end{array} \right.$

and of course :

$\left\{ \begin{array}{l} N \longrightarrow telescope \\ \langle N\ number \rangle = singular \end{array} \right.$

$\left\{ \begin{array}{l} N \longrightarrow telescopes \\ \langle N\ number \rangle = plural \end{array} \right.$

We now are able to obtain the NPs “a telescope”, “the telescope”, “telescopes” and “the telescopes”¹.

3.0.4 Generalisation

Another possible notation consists in writing rules in which symbols are either atoms (for the terminals) or feature structures (for the non terminals). If so, the variables appearing in the feature structures will be global to the rules. Our fragment then becomes :

$\left[\begin{array}{l} cat = NP \\ number = \square_1 \end{array} \right]$	$\longrightarrow$	$\left[\begin{array}{l} cat = D \\ number = \square_1 \end{array} \right]$	$\left[\begin{array}{l} cat = N \\ number = \square_1 \end{array} \right]$
$\left[\begin{array}{l} cat = D \end{array} \right]$	$\longrightarrow$	the	
$\left[\begin{array}{l} cat = D \\ number = singular \end{array} \right]$	$\longrightarrow$	a	
$\left[\begin{array}{l} cat = D \\ number = plural \end{array} \right]$	$\longrightarrow$	ϵ	
$\left[\begin{array}{l} cat = N \\ number = singular \end{array} \right]$	$\longrightarrow$	telescope	
$\left[\begin{array}{l} cat = N \\ number = plural \end{array} \right]$	$\longrightarrow$	telescopes	

This notation is more general than the one we had at previous section. In our example, we distinguished the feature “cat” : each feature structure appearing in the rules has a “cat” feature with

¹In our example, no category appeared more than once in a rule. If it had been the case, we would index the categories in order to distinguish their occurrences

an atomic value. From a formal point of view, this is not mandatory : we could conceive unification grammars without any clear context free skeleton.

3.0.5 The intersection grammar

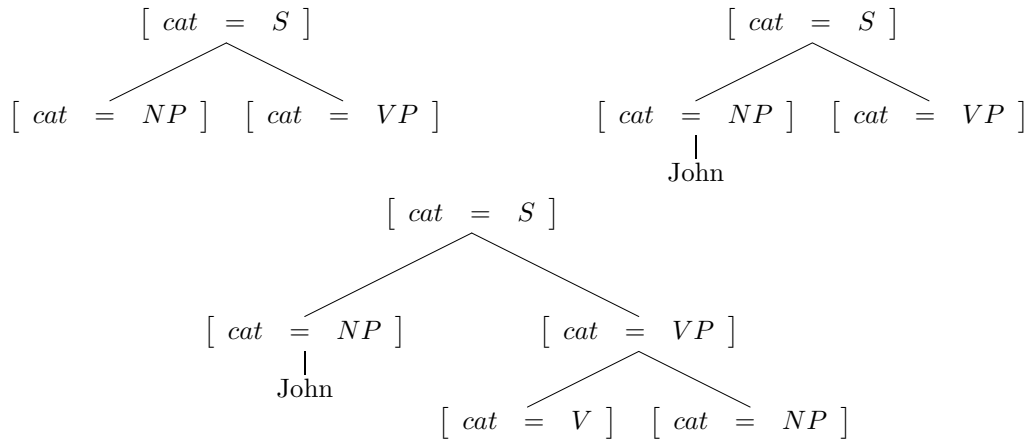
The rules of the game

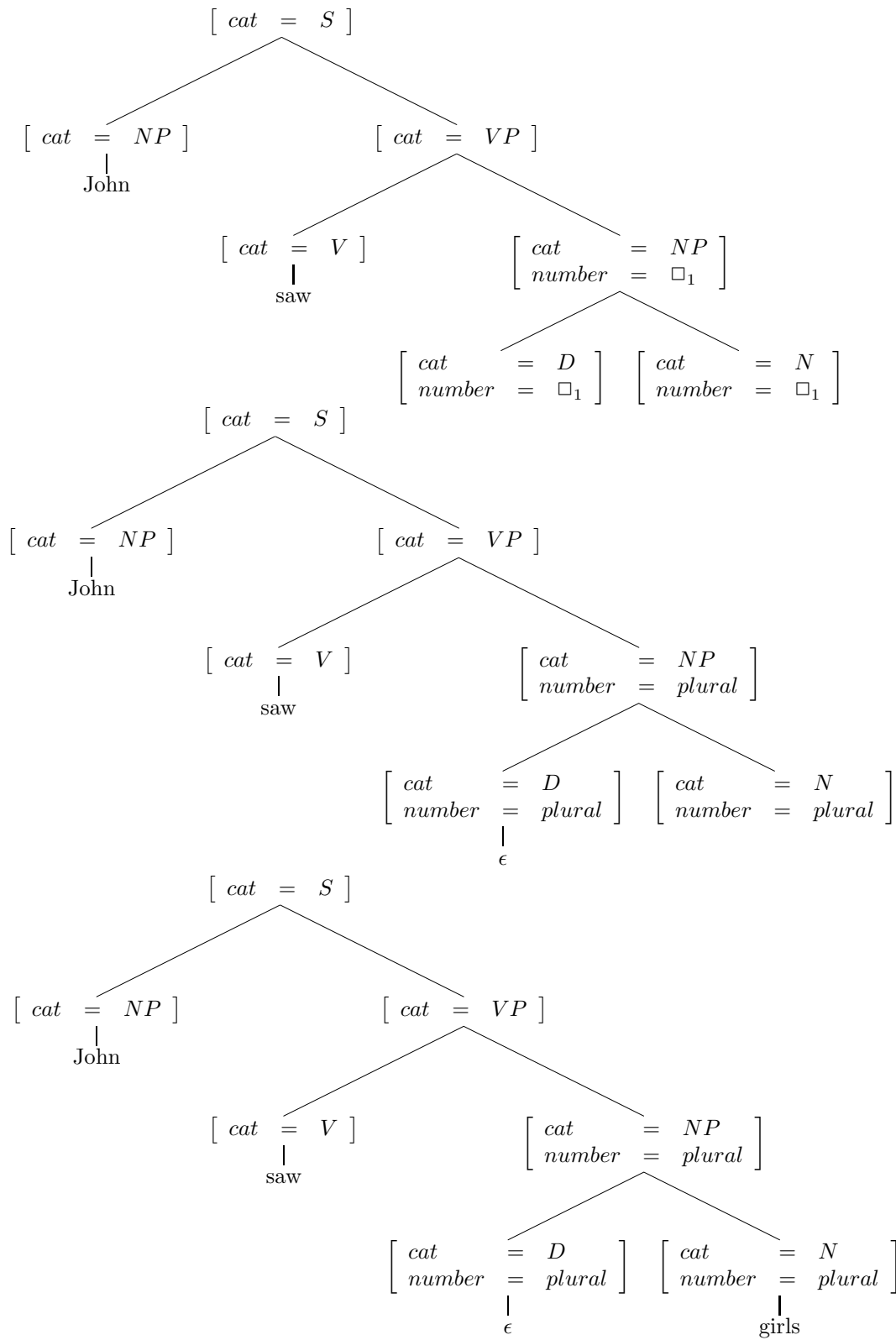
As the non terminal of the grammar are now feature structures, the notion of derivation has to be changed. In standard context free grammars we relied on symbols identity : a non terminal appearing as a leaf in a tree could be replaced by a right part of a rule that defined it. Now, a feature structure appearing as a leaf may be replaced as well but modulo unification.

In order to illustrate that point, we consider the following grammar :

$[cat = S]$	$\longrightarrow$	$[cat = NP]$	$[cat = VP]$
$[cat = VP]$	$\longrightarrow$	$[cat = V]$	$[cat = NP]$
$[cat = VP]$	$\longrightarrow$	$[cat = V]$	$[cat = NP]$ $[cat = PP]$
$[cat = NP]$ $[number = singular]$	$\longrightarrow$	John	
$[cat = NP]$ $[number = \square_1]$	$\longrightarrow$	$[cat = D]$ $[number = \square_1]$	$[cat = N]$ $[number = \square_1]$
$[cat = NP]$ $[number = \square_1]$	$\longrightarrow$	$[cat = D]$ $[number = \square_1]$	$[cat = N]$ $[number = \square_1]$ $[cat = PP]$
$[cat = D]$	$\longrightarrow$	the	
$[cat = D]$ $[number = singular]$	$\longrightarrow$	a	
$[cat = D]$ $[number = plural]$	$\longrightarrow$	ϵ	
$[cat = N]$ $[number = singular]$	$\longrightarrow$	telescope	
$[cat = N]$ $[number = plural]$	$\longrightarrow$	telescopes	
$[cat = N]$ $[number = singular]$	$\longrightarrow$	girl	
$[cat = N]$ $[number = plural]$	$\longrightarrow$	girls	
$[cat = PP]$	$\longrightarrow$	$[cat = P]$ $[cat = NP]$	
$[cat = P]$	$\longrightarrow$	with	

An example derivation is then :





The intersection grammar

We can produce the results exactly as we did it in the context free case. Namely decorate the grammar rules with all possible indexes. As in the context free case, in order to get the results, we will have to

make the grammar generate resulting trees. The problem now is that this generation phase as to be done modulo unification.

In our preceeding example, this corresponds to having a grammar containing (among a lot other rules) :

$\left[\begin{array}{l} cat = S \end{array} \right]_{0,3}$	$\longrightarrow$	$\left[cat = NP \right]_{0,1} \left[cat = VP \right]_{1,3}$
$\left[\begin{array}{l} cat = NP \end{array} \right]_{0,1}$	$\longrightarrow$	John _{0,1}
$\left[\begin{array}{l} cat = VP \end{array} \right]_{1,3}$	$\longrightarrow$	$\left[cat = V \right]_{1,2} \left[cat = NP \right]_{2,3}$
$\left[\begin{array}{l} cat = V \end{array} \right]_{1,2}$	$\longrightarrow$	saw _{1,2}
$\left[\begin{array}{l} cat = NP \\ number = \square_1 \end{array} \right]_{2,3}$	$\longrightarrow$	$\left[\begin{array}{l} cat = D \\ number = \square_1 \end{array} \right]_{2,2} \left[\begin{array}{l} cat = N \\ number = \square_1 \end{array} \right]_{2,3}$
$\left[\begin{array}{l} cat = D \\ number = plural \end{array} \right]_{2,2}$	$\longrightarrow$	$\epsilon_{2,2}$
$\left[\begin{array}{l} cat = N \\ number = plural \end{array} \right]_{2,3}$	$\longrightarrow$	girls _{2,3}

The notion of grammar of the intersection makes sense, what we get is of course a unification grammar. We now face the same problem we had with pure CF grammars, that is deciding wether this grammar generates something².

3.0.6 Expressiveness

Unification grammars are strictly more expressive than context free grammars. In a unification grammar, the context around a non terminal influences the possible rewriting of that non terminal. What actually happens is that unification grammars are as expressive as Turing Machines, to state it simply, they are powerfull enough to encode any computer program. The simplest way to convince oneself is to compare unification grammars with prolog programs. A prolog program is a list of clauses of the form³ :

predicate ::= *list_of_predicates*

or of the form

predicate

together with a goal. One can easilly encode this prolog program as a(n almost equivalent) unification grammar⁴.

This has a consequence : the recognition problem for unification grammars in general is undecidable. There are however subclasses that are perfectly decidable :

- unification grammar in which feature structures are atomic (the value of an attribute is either a variable or a symbol) are decidable. From a theoretical point of view, such grammars could be rewritten as context free grammars by naming differently all possible feature structures and there is a finite number of them.
- more generally, if feature structure cannot become deeper than some fixed depth, the recognition problem is decidable.

3.0.7 Parsing algorithms

The two parsing algorithm we saw for context free grammars can be adapted to cope with unification grammars. The difference however is that we cannot test the identity of symbols : we have to work modulo unification of feature structures. We will adopt the same strategy we took for context free grammars. Our first version of the CKY algorithm directly computes a unification grammar.

²Note that ϵ in rules have to be labelled with first index equals second one

³A good reference concerning prolog is the course taught by Kristina Striegnitz

⁴There are differences however : the order of the clauses is important in Prolog whereas it is not in a unification grammar. Moreover Prolog also has the “cut” operator and predefined predicates.

CKY revisited

We can adopt the same analysis schema we had for context free grammars :

initialisation

For each terminal appearing in the input stream, produce a grammar rule such as :

$$w_{i,i+1} \longrightarrow w$$

for ϵ rules, produce all $E_{i,i} \longrightarrow \epsilon$.

loop So long as we can produce new grammar rules, do produce a grammar rule such as $C_{i,j} \longrightarrow A_{i,k} B_{k,l} \dots K_{m,j}$ if and only if there exists rules with left $A'_{i,k}, B'_{k,l}, \dots K'_{m,j}$ so that they unify in a rule of the original grammar of the form $A' \longrightarrow B C \dots K$

On our new favorite example (i.e. “John saw girls with the telescope”), this gives us :

$\text{John}_{0,1}$	$\longrightarrow$	John
$\text{saw}_{1,2}$	$\longrightarrow$	saw
$\epsilon_{2,2}$	$\longrightarrow$	ϵ
$\text{girls}_{2,3}$	$\longrightarrow$	girls
$\text{with}_{3,4}$	$\longrightarrow$	with
$\text{the}_{4,5}$	$\longrightarrow$	the
$\text{telescope}_{5,6}$	$\longrightarrow$	telescope
$\left[\begin{array}{l} \text{cat} = NP \end{array} \right]_{0,1}$	$\longrightarrow$	$\text{John}_{0,1}$
$\left[\begin{array}{l} \text{cat} = V \end{array} \right]_{1,2}$	$\longrightarrow$	$\text{saw}_{1,2}$
$\left[\begin{array}{l} \text{cat} = N \\ \text{number} = plural \end{array} \right]_{2,3}$	$\longrightarrow$	$\text{girls}_{2,3}$
$\left[\begin{array}{l} \text{cat} = P \end{array} \right]_{3,4}$	$\longrightarrow$	$\text{with}_{3,4}$
$\left[\begin{array}{l} \text{cat} = D \end{array} \right]_{4,5}$	$\longrightarrow$	$\text{the}_{4,5}$
$\left[\begin{array}{l} \text{cat} = N \\ \text{number} = singular \end{array} \right]_{5,6}$	$\longrightarrow$	$\text{telescope}_{5,6}$
$\left[\begin{array}{l} \text{cat} = NP \\ \text{number} = plural \end{array} \right]_{2,3}$	$\longrightarrow$	$\epsilon_{2,2} \left[\begin{array}{l} \text{cat} = N \\ \text{number} = plural \end{array} \right]_{2,3}$
$\left[\begin{array}{l} \text{cat} = NP \\ \text{number} = singular \end{array} \right]_{4,6}$	$\longrightarrow$	$\left[\begin{array}{l} \text{cat} = D \\ \text{number} = singular \end{array} \right]_{4,5} \left[\begin{array}{l} \text{cat} = N \\ \text{number} = singular \end{array} \right]_{5,6}$
$\left[\text{cat} = PP \right]_{3,6}$	$\longrightarrow$	$\left[\text{cat} = P \right]_{3,4} \left[\begin{array}{l} \text{cat} = NP \\ \text{number} = singular \end{array} \right]_{4,6}$
$\left[\text{cat} = VP \right]_{1,3}$	$\longrightarrow$	$\left[\text{cat} = V \right]_{1,2} \left[\begin{array}{l} \text{cat} = NP \\ \text{number} = plural \end{array} \right]_{2,3}$
$\left[\text{cat} = S \right]_{0,3}$	$\longrightarrow$	$\left[\text{cat} = NP \right]_{0,1} \left[\text{cat} = VP \right]_{1,3}$
$\left[\text{cat} = VP \right]_{1,6}$	$\longrightarrow$	$\left[\text{cat} = V \right]_{1,2} \left[\begin{array}{l} \text{cat} = NP \\ \text{number} = plural \end{array} \right]_{2,3} \left[\text{cat} = PP \right]_{3,6}$
$\left[\text{cat} = NP \right]_{2,6}$	$\longrightarrow$	$\left[\begin{array}{l} \text{cat} = NP \\ \text{number} = plural \end{array} \right]_{2,3} \left[\text{cat} = PP \right]_{3,6}$

There is a big difference between unification grammars and CF-grammars. In the CF case, we could argue that the equivalent version of CKY would not loop indefinitely because the intersection grammars was finite. It is not the case anymore : we do not have a finite number of non-terminals. The problem of looping can therefore happen in the case of an infinitely ambiguous grammar⁵.

In this version of the algorithm, this directly corresponds to deciding whether a newly produced rule is actually new. For instance, suppose we already produced the rule :

$$\left[\text{cat} = NP \right]_{1,3} \longrightarrow \left[\text{cat} = Det \right]_{1,2} \left[\text{cat} = N \right]_{2,3}$$

and that we now produce the rule :

$$\left[\begin{array}{l} \text{cat} = NP \\ \text{gender} = masculine \end{array} \right]_{1,3} \longrightarrow \left[\begin{array}{l} \text{cat} = Det \\ \text{gender} = masculine \end{array} \right]_{1,2} \left[\begin{array}{l} \text{cat} = N \\ \text{gender} = masculine \end{array} \right]_{2,3}$$

⁵The problem is that terms may grow bigger and bigger

it seems quite natural to consider that the latter is not a “new rule” as we already have a more general rule. More generally this technique may avoid some of the possible loops : there are cases where we would continuously produce new rules with bigger terms that are less general than terms we already have. In such cases, detecting that a rule is subsumed by another avoids looping.

3.0.8 CKY for unification grammars

The idea is exactly the same as it was for context free grammars : what is important when we look at the previous example is only the fact that some feature structure rewrites as some sequence of terminals of the input string. We will thus have items of the form : $\langle i, \text{some_feature_structure}, j \rangle$ that correspond to exactly this. The algorithm is then described by the following inference rules :

$$\frac{\langle i, w, i \rangle}{w}$$

if w appears in the input string between i and $i + 1$

$$\frac{\langle i, A, j \rangle \quad \langle j, B, k \rangle}{\langle i, C, k \rangle}$$

if there is a grammar rule $C' \rightarrow A' B'$ in the grammar and if A and B can be unified in that rule with A' and B' and that C corresponds to C' after these unifications have been done.

An item will be considered as new, only if no more general item was already produced.

The generalized CKY algorithm for grammars (for rules of length greater than 2) can also be adapted exactly with the same idea.

3.0.9 Earley for unification grammars

The Earley algorithm can be also be adapted for unification grammars. A new problem can however appear there. The Earley algorithm aims at reducing the number of possible grammars rules in the result. This corresponds to the predict inference rule :

$$\frac{\langle i, A \rightarrow \alpha \bullet B\beta, j \rangle}{\langle j, B \rightarrow \bullet\gamma, j \rangle}$$

This rule aims at authorizing an attempt to recognize B from the index j only in case B actually can occur in some context. If we extend this prediction rule for unification grammars, it seems at first that this inference rule should be considered modulo unification. This means that we can predict $\langle j, B' \rightarrow \bullet\gamma', j \rangle$ with B' and γ' the result of the unification of B in the grammar rule. As unification is involved, if it succeeds, it produces terms that are more specific than those unified, we will therefore be very predictive.

The problem is that such predictions may produce loops, simply because terms may grow without subsuming each other. The paradox in this case is that these predictions are not necessary for the correctness of the algorithm. There are therefore two possible solutions :

- predict the grammar rule as it is (that is testing that the rule is applicable (by unification) but predict an item with the rule as it is in the grammar.
- predict some generalisation of the result of the unification [Shi85]

3.0.10 Offline parsability

General unification grammars are a formalism so general that they have the expressive power of Turing Machines. Intuitively, any prolog program (without cut) can be represented quite straightforwardly as an equivalent unification grammar (the goal is the axiom).

There are subclasses of unification grammars for which the recognition problem is decidable. The first proposed is so-called “off-line parsable grammars” [PW83].

A grammar is off-line parsable if its context-free skeleton is finitely ambiguous. On such a grammar, the recognition problem is decidable and the parsing algorithms terminate.

3.0.11 Extraction of the results

The results for unification grammars can be extracted exactly as they were for context free grammars. The only difference is that as the algorithms only store new items in the chart and that newness in this context corresponds to the fact that no more general item was already produced, some unification still need to be performed when extracting the results. Simply put, this means that as the result is a unification grammar, one has to use it as such and respect the rewriting game of such grammars. However, the parsing algorithms already verified that the sentence is generated by the grammar therefore, no unification can fail when extracting the results. This corresponds to the notion of interaction free grammar proposed by [Dym97].

Chapter 4

Parsing with polarities

4.1 Introduction

In unification grammars, syntactic composition is performed by a mechanism of unification between feature structures. For instance, let us parse the sentence *John loves Mary* with a lexicalized unification grammar. Here is the part of the lexicon used for parsing the sentence:

$$\begin{array}{lcl}
 \textit{John} : & \left[\begin{array}{l} \textit{cat} : n \\ \textit{phon} : \textit{"John"} \\ \textit{agr} : \left[\begin{array}{l} \textit{num} : sg \end{array} \right] \end{array} \right. \\
 \\
 \textit{loves} : & \left[\begin{array}{l} \textit{cat} : s \\ \textit{phon} : X.\textit{"loves"}.Y \\ \textit{head} : \left[\begin{array}{l} \textit{cat} : v \\ \textit{phon} : \textit{"loves"} \\ \textit{mode} : ind \\ \textit{tense} : pres \end{array} \right] \\ \textit{subj} : \left[\begin{array}{l} \textit{cat} : n \\ \textit{phon} : X \\ \textit{agr} : \left[\begin{array}{l} \textit{pers} : 3 \\ \textit{num} : sg \end{array} \right] \end{array} \right] \\ \textit{obj} : \left[\begin{array}{l} \textit{cat} : n \\ \textit{phon} : Y \end{array} \right] \end{array} \right. \\
 \\
 \textit{Mary} : & \left[\begin{array}{l} \textit{cat} : n \\ \textit{phon} : \textit{"Mary"} \\ \textit{agr} : \left[\begin{array}{l} \textit{num} : sg \end{array} \right] \end{array} \right.
 \end{array}$$

Atomic feature values are elements of domains, each domain being associated with a feature. Generally, domains are not structured, except that of the *phon* feature which is a monoid¹.

In unification grammars, the sharing of values between features is expressed by a mechanism of co-indexation. Here, we restrict sharing to atomic values and we express it by means of local variables. In the example above, we use two variables *X* and *Y* representing phonological terms².

By unifying the values of the features *subj* and *obj* with the structures respectively associated with *John* and *Mary*, we obtain the following feature structure associated with the sentence *John loves Mary*:

¹A monoid is a set with a binary operation, denoted “.”, that has two properties: it is associative, that is for any elements *x*, *y*, *z*, (*x.y*).*z* = *x*.(*y.z*), and it has an identity element, that is an element *e* such that *e.x*=*x.e* for any element *x*.

²The names of variables do not matter: these variables are local to their feature structures.

$cat :$	s										
$phon :$	$"John\ loves\ Mary"$										
$head :$	<table> <tr><td>$cat :$</td><td>v</td></tr> <tr><td>$phon :$</td><td>$"loves"$</td></tr> <tr><td>$mode :$</td><td>ind</td></tr> <tr><td>$tense :$</td><td>$pres$</td></tr> </table>	$cat :$	v	$phon :$	$"loves"$	$mode :$	ind	$tense :$	$pres$		
$cat :$	v										
$phon :$	$"loves"$										
$mode :$	ind										
$tense :$	$pres$										
$subj :$	<table> <tr><td>$cat :$</td><td>n</td></tr> <tr><td>$phon :$</td><td>$"John"$</td></tr> <tr><td>$agr :$</td><td> <table> <tr><td>$pers :$</td><td>3</td></tr> <tr><td>$num :$</td><td>sg</td></tr> </table> </td></tr> </table>	$cat :$	n	$phon :$	$"John"$	$agr :$	<table> <tr><td>$pers :$</td><td>3</td></tr> <tr><td>$num :$</td><td>sg</td></tr> </table>	$pers :$	3	$num :$	sg
$cat :$	n										
$phon :$	$"John"$										
$agr :$	<table> <tr><td>$pers :$</td><td>3</td></tr> <tr><td>$num :$</td><td>sg</td></tr> </table>	$pers :$	3	$num :$	sg						
$pers :$	3										
$num :$	sg										
$obj :$	<table> <tr><td>$cat :$</td><td>n</td></tr> <tr><td>$phon :$</td><td>$"Mary"$</td></tr> <tr><td>$agr :$</td><td> <table> <tr><td>$num :$</td><td>sg</td></tr> </table> </td></tr> </table>	$cat :$	n	$phon :$	$"Mary"$	$agr :$	<table> <tr><td>$num :$</td><td>sg</td></tr> </table>	$num :$	sg		
$cat :$	n										
$phon :$	$"Mary"$										
$agr :$	<table> <tr><td>$num :$</td><td>sg</td></tr> </table>	$num :$	sg								
$num :$	sg										

Feature structures can be viewed as directed acyclic graphs (DAGs), that are graphs without cycles : features are represented by edges labelled with feature names and values by vertices labelled with these values when they are atomic. Since we restrict sharing to atomic values, DAGs reduce to trees. Figure 4.1 shows the lexical entries of the words *John*, *loves*, *Mary*, which were presented previously in the shape of feature structures and which are presented now in the shape of trees. On the figure, thick

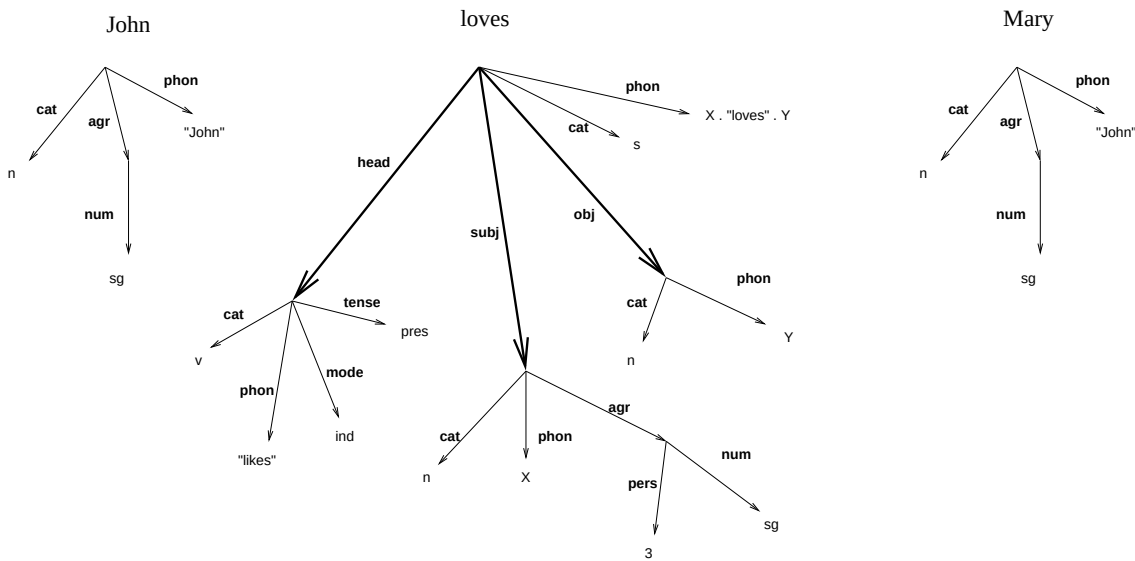


Figure 4.1: Trees representing the lexical entries of the words *John*, *loves*, *Mary*

edges represent syntactic dependencies between constituents and their immediate components whereas thin edges represent membership relations between constituents and features or between features and sub-features.

With trees, syntactic composition, which was previously performed as unification between feature structures, now appears as partial superposition of trees. With respect to graph superposition, tree superposition includes an additional constraint: tree characteristics must be preserved, in other words no cycles must be created, every node has exactly one mother node except one node, the root, which has no mother node. Another constraint stems from the specific use of trees: for every node, the relationship between features and their values is functional: there is one edge at most that starts from a given node and which is labelled with a given feature. For instance, if we try to superpose the roots of the trees associated with *John* and *loves*, the previous constraint entails the superposition of the two edges labelled with *cat*, hence the superposition of their targets and the failure of the operation because these targets are labelled with two distinct values *n* and *s*. In our example, there is only one way of superposing the three trees associated with the words *John*, *loves*, *Mary* to get a syntactic representation of the sentence *John*

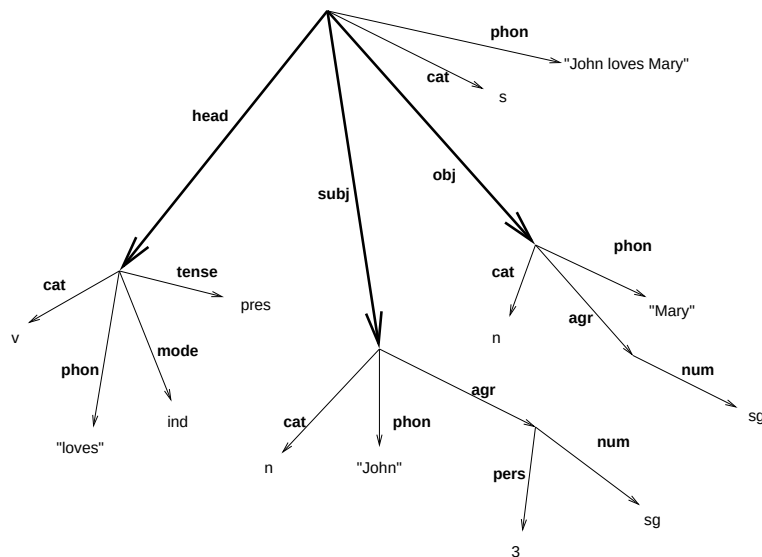


Figure 4.2: Tree representing the syntactic structure of the sentence *John loves Mary*.

loves Mary in the shape of the tree given by figure 4.2. The use of unification as a unique mechanism of syntactic composition presents a major default: this mechanism includes no indication about the saturation of the composed syntactic structures, that is to which extent they are complete syntactic representations of a sentences. The fact that the root of a structure has a *cat* feature with the value *s* and the absence of non-instantiated variables are necessary conditions, but not sufficient conditions.

In order to express this property, we propose to use *polarities*. This idea is old and it is at the heart of the Tesnière's vision [Tes34]: he considers a sentence as a molecule and its words as the atoms that gather for constituting this molecule; every atom is equipped with a valence which allows it to interact with other atoms. The notion of polarity underlies Categorical Grammars but it is used explicitly only recently in formal grammars [Nas95, MK00, DT99, Per00, Per02, Per04, Kah04].

In a first approach, we use a system of four polarities: $+$, $-$, $=$ and $\overset{\circ}{=}$ with an operation of addition, denoted $\sqcup$. This operation is associative, commutative and $=$ is its identity element. Except for $=$, $\sqcup$ is only defined for two sums: $+\sqcup-$ and $-\sqcup+$, which are both equal to $\overset{\circ}{=}$. In trees expressing syntactic structures, nodes that represent constituents (roots and targets of syntactic dependency edges) are labelled with polarities with the following meaning: a constituent labelled with the negative polarity $-$ represents an expected constituent whereas a constituent labelled with the positive polarity $+$ represents an available resource and both can unify for building a constituent labelled with the polarity $\overset{\circ}{=}$, which cannot interact with any other constituent³. Nodes that are labelled with the polarity $=$ do not behave as resources and can be superposed with any other node any number of times; they represent constituents or features indifferently.

Now, with the notion of polarity, we can differentiate saturated syntactic structures from unsaturated syntactic structures: a saturated structure has all its polarities neutral or saturated whereas an unsaturated structure keeps positive or negative polarities, which express its ability to interact with other structures.

Let us take again our three lexical entries in the shape of trees and let us equip them with polarities in order to express their interaction ability. A transitive verb expects two noun phrases, its subject and its object, for building a sentence; therefore, in the syntactic tree that anchors such a verb, we put the polarity $-$ on the nodes representing the expected subject and object and we put the polarity $+$ on the node representing the resulting sentence. The two proper nouns *John* and *Mary* behave as available noun phrases, thus we put the polarity $+$ on the corresponding nodes. Other nodes are equipped with the

³In the formalism of Interaction Grammars [Per00, Per02, Per04], polarities are attached to features and not to constituents.

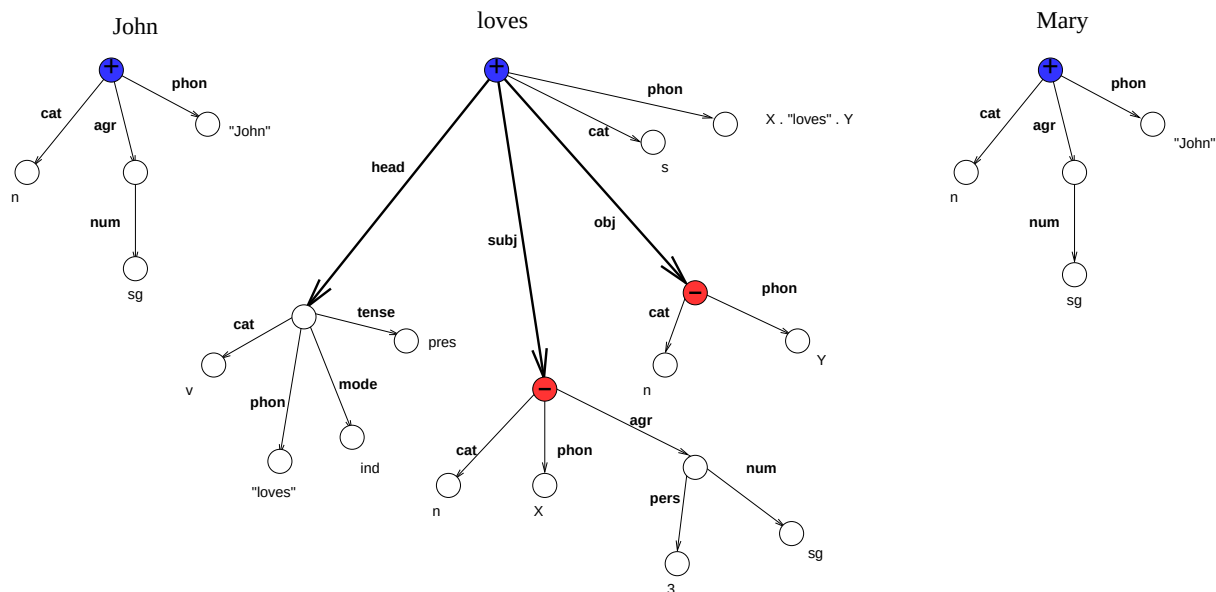


Figure 4.3: Polarized trees representing the lexical entries of the words *John*, *loves*, *Mary*

polarity =. To summarize, we obtain the polarized trees of figure 4.3.

The introduction of polarities add new constraints to syntactic composition, which are used for guiding this process: two nodes are superposable if they carry opposite polarities or if one is neutral and complete trees must be saturated, that is without positive and negative nodes. In our example, these new constraints entail the merging of the two negatives nodes with the positive roots of the trees anchored by *John* and *Mary*. There are two possibilities of doing this, which correspond to the two sentences *John loves Mary* and *Mary loves John*. If we choose the first alternative, we obtain a tree, which is not totally saturated because its roots can interact for instance with a tree representing the phrase *Peter says* for building the syntactic representation of the sentence *Peter says John loves Mary*. In order to close parsing, we have to introduce a polarized tree, which plays the role of an axiom as in other formalisms, and which represents the parsing request: we expects a sentence as the output of the parsing process, thus the corresponding tree is constituted of a negative node representing the expected sentence with a unique *cat* feature pointing the value *s*. This tree is represented on figure 4.4 with the tree coming from the composition of the sentence *John loves Mary*. The two trees are superposed for building the neutral tree of figure 4.5, which ends the parsing process.

4.2 Polarized Unification Grammars

In this section, we propose a formalization of the notion of polarity within unification grammars, defining the generic formalism of Polarized Unification Grammars. In the following, we refer to the formalism by the acronym PUG and to a particular polarized unification grammar by PU grammar. PUG elaborates on the proposal of [Kah04] with a more restricted system of polarities.

For this, we need a signature, which is constituted of a set *Dep* of names of syntactic dependencies and a set *Feat* of feature names. For instance, in the example of section 4.1, $Dep = \{head, obj, subj, \dots\}$ and $Feat = \{agr, cat, phon, num, pers, mode, tense, \dots\}$.

The set *Feat* is divided into compound features, like *agr*, and atomic features, like *cat*, *phon*, *num*, *pers*, *mode*, *tense*. Every feature *f* is equipped with a domain of values D_f . If *f* is atomic, D_f generally reduces to a set of atoms; for instance, $D_{num} = \{sg, pl\}$. D_{phon} is an exception: it is an infinite set including all finite strings of words for the concerned language; moreover, it is structured as a monoid by the operation of concatenation between strings. The possible values of a feature *f* are either constants coming from D_f or variables, except for *phon*: *phon* values are concatenations of constant words and

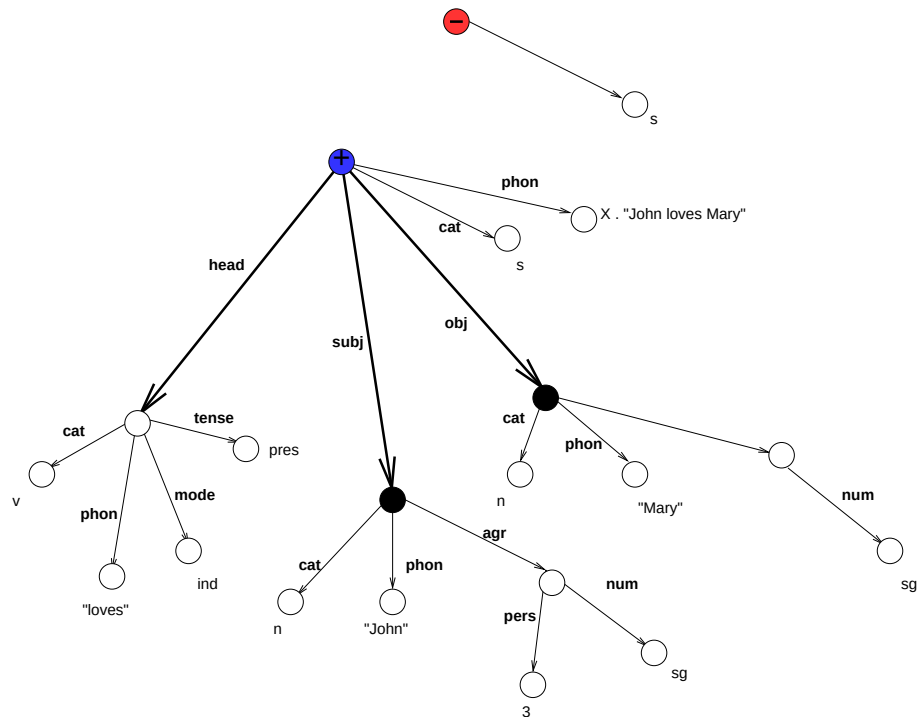


Figure 4.4: Polarized trees representing the sentence *John loves Mary* and the output of parsing

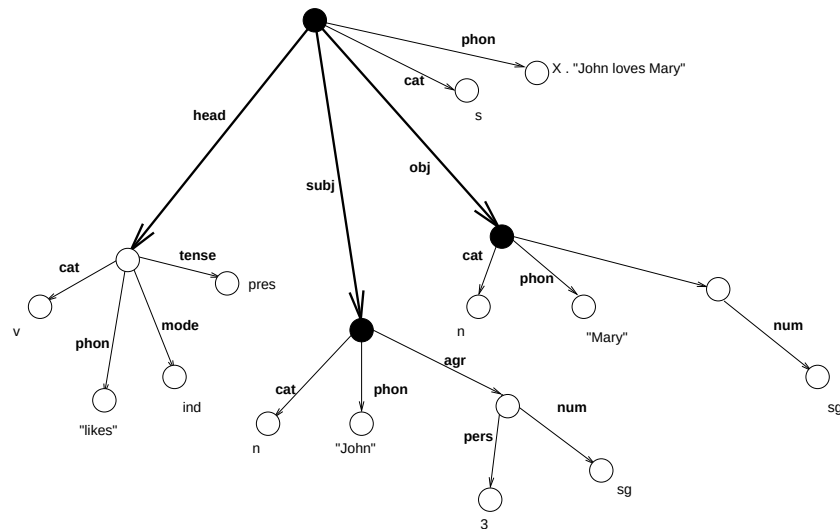


Figure 4.5: Neutral tree representing the parse tree of the sentence *John loves Mary*

string variables; for instance, $X.\text{"loves"}.Y$ is a possible value of the *phon* feature.

The syntactic structures manipulated in PUG are *polarized syntactic trees*. The operation that allows them to be composed is *tree superposition* and the structures representing the syntax of acceptable sentences are *saturated syntactic trees*. Now, we define these notions precisely.

Definition 4.2.1 *A polarized syntactic tree is a tree every node of which is labelled with a polarity $+$, $-$, $=$ or $\doteq$. Nodes are either feature values or constituents. Edges are labelled with features names when their target is a feature value, and possibly with syntactic functions when their target is a constituent. In this case, their source is also a constituent. Edges labelled with atomic feature names have their target labelled with a consistent feature value.*

All labels of the edges starting from the same source are different.

Figures 4.3, 4.4 and 4.5 give some examples of polarized syntactic trees.

Polarized syntactic trees are composed together by *superposition*.

Definition 4.2.2 *A superposition of two polarized syntactic trees is a partial superposition of trees, where the meaning should be immediate, with the following operations on polarities and labels:*

- *nodes that are superposed must bear polarities for which the sum exists and this sum is attached to the resulting node;*
- *other labels of superposed edges or nodes are identified⁴ except phon values which are unified according to their particular structure⁵.*

We go from figure 4.3 to the right tree of figure 4.4 by two successive superpositions and from figure 4.4 to figure 4.5 by one last superposition.

Polarities allow the distinction between saturated and unsaturated syntactic structures.

Definition 4.2.3 *Saturated syntactic structures are syntactic structures without positive and negative polarities.*

Syntactic structures of figure 4.3 and 4.4 are unsaturated whereas the syntactic structure of figure 4.5 is saturated. According to our notion of syntactic composition, saturated syntactic structures cannot be composed with other structures.

A particular PU grammar G is defined by a set of polarized syntactic trees, which constitute its elementary trees and by an axiom, which is a polarized tree representing the expected sentence as the result of parsing. A polarized tree derived from the grammar G is obtained by superposing elementary trees a finite number of times. Superposition can include the axiom but once at most. The language $L(G)$ generated by the grammar G is the set of *phon* values of roots of saturated trees that are derived by using the axiom once.

4.3 The example of Categorical Grammars

The only well-known formalism that uses the resource sensitivity of natural languages for driving syntactic composition is Categorical Grammars (CG). Even if the notion of polarity is not explicit in CG, this underlies the whole formalism. In this section, we show that CG, in their original presentation, can be represented as a particular restriction of PUG.

⁴We suppose that the sets of variables of the two structures to superpose are disjoint, which corresponds to the locality of variables.

⁵Let $v_0.X_1.v_1.X_2 \dots v_{n-1}.X_n.v_n$ and $w_0.Y_1.w_1.Y_2 \dots w_{p-1}.Y_p.w_p$ be two *phon* values to unify, where $v_0, v_1, \dots, v_n, w_0, w_1, \dots, w_p$ are strings of words and $X_1, X_2, \dots, X_n, Y_1, Y_2, \dots, Y_p$ are variables. The unification of the two values consists in searching for a substitution $\{X_1 \rightarrow t_1, X_2 \rightarrow t_2, \dots, X_n \rightarrow t_n, Y_1 \rightarrow u_1, Y_2 \rightarrow u_2, \dots, Y_p \rightarrow u_p\}$ such that $t_1, t_2, \dots, t_n, u_1, u_2, \dots, u_p$ are *phon* values and $v_0.t_1.v_1.t_2 \dots v_{n-1}.t_n.v_n = w_0.u_1.w_1.u_2 \dots w_{p-1}.u_p.w_p$.

$$\begin{aligned}
\text{"John"}: & \quad np \\
\text{"Mary"}: & \quad np \\
\text{"loves"}: & \quad (np \backslash s) / np \\
\text{"passionately"}: & \quad (np \backslash s) \backslash (np \backslash s)
\end{aligned}$$

Figure 4.6: The toy AB grammar G_0

$$\begin{array}{c}
\frac{\text{"loves"}: (np \backslash s) / np \quad \text{"Mary"}: np}{\text{"loves Mary"}: np \backslash s} (//) \\
\frac{\text{"John"}: np \quad \frac{\text{"loves Mary"}: np \backslash s \quad \text{"passionately"}: (np \backslash s) \backslash (np \backslash s)}{\text{"loves Mary passionately"}: np \backslash s} (\wedge) \\
\hline
\text{"John loves Mary passionately"}: s \quad (\wedge)
\end{array}$$

Figure 4.7: Derivation of the sentence John loves Mary passionately with the grammar G_0

4.3.1 AB grammars

CG are the first attempt at formalizing the syntax of natural languages. In the 1930s, Adjukiewicz defined a calculus of fractions for representing well-formed logical propositions [Adj35]. Then, in the 50s, Bar-Hillel refined this calculus for representing the syntax of sentences from natural languages [BH53]. The syntactic types of constituents are built recursively from a set of atomic types by means of two division operators: a left operator, denoted " $\backslash$ ", and a right operator, denoted " $/$ ". Their meaning is given by the following rules of backward and forward application:

$$\frac{w_1 : B \quad w_2 : B \backslash A}{w_1.w_2 : A} (\backslash) \qquad \frac{w_1 : A/B \quad w_2 : B}{w_1.w_2 : A} (/)$$

Backward application rule means that an expression w_2 with the type $B \backslash A$ expects an expression w_1 with the type B on its left to be concatenated with it in order to build an expression $w_1.w_2$ with the type A .

Forward application rule means that an expression w_1 with the type A/B expects an expression w_2 with the type B on its right to be concatenated with it in order to build an expression $w_1.w_2$ with the type A .

By iterating backward and forward applications, we build derivation trees. If a derivation tree has $w_1.w_2 \dots w_n : C$ as its root with the sequence $w_1 : H_1, w_2 : H_2, \dots, w_n : H_n$ as its leaves, we denote the corresponding derivation: $w_1 : H_1, w_2 : H_2, \dots, w_n : H_n \xrightarrow{*} w_1.w_2 \dots w_n : C$.

We refer to the formalism of Adjukiewicz Bar-Hillel Grammars by the acronym ABG and to a particular grammar expressed in ABG by AB grammar. An AB grammar G is defined by its lexicon, which links every word of the language with a set of syntactic types. We denote $G(w)$ the set of syntactic types associated to the word w by the grammar G . The language $L(G)$ generated by the grammar G is the set of expressions of the form $w_1.w_2 \dots w_n$ for which there exists a sequence $H_1, \dots, H_n$ of syntactic types such that:

1. for any $1 \leq i \leq n$, $H_i \in G(w_i)$,
2. $w_1 : H_1, w_2 : H_2, \dots, w_n : H_n \xrightarrow{*} w_1.w_2 \dots w_n : S$ with S being the type *sentence*.

As an example, consider the small toy grammar G_0 given by figure 4.6. The derivation tree given by figure 4.7 justifies the derivation "*John*" : np , "*loves*" : $(np \backslash s) / np$, "*Mary*" : np , "*passionately*" : $(np \backslash s) \backslash (np \backslash s) \xrightarrow{*}$ "*John loves Mary passionately*" : s . The language $L(G_0)$ is constituted of the four following sentences: "*John loves Mary*", "*Mary loves John*", "*John loves Mary passionately*", "*Mary loves John passionately*".



Figure 4.8: Tree initialization for the computation of $t^-(w : A/B)$ and $t^+(w : A/B)$

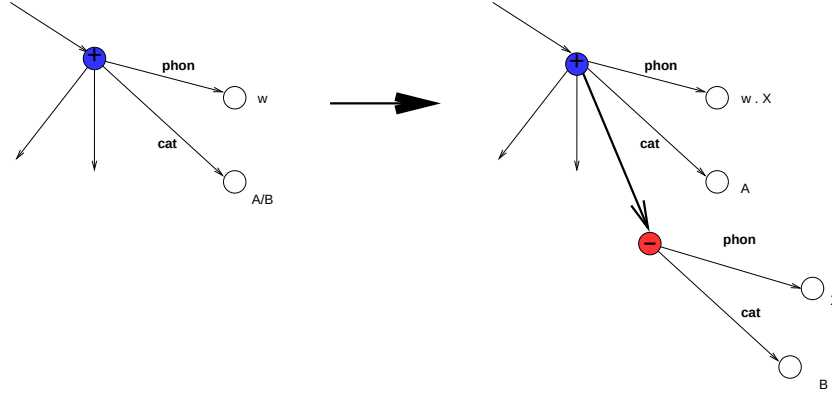


Figure 4.9: Splitting rule for positive nodes

4.3.2 Translation into the PUG formalism

In this subsection, we will show that AB grammars can be considered as particular PU grammars. For this, we define two functions: a positive function that translates any expression w with the syntactic type A from ABG into a polarized tree $t^+(w : A)$ of PUG and a negative function that translates any expression with the syntactic type A from an ABG into a polarized tree $t^-(w : A)$ of PUG.

The functions t^+ and t^- are defined by means of a tree rewriting system⁶. The computation of $t^-(w : A/B)$ and $t^+(w : A/B)$ for any expression w of the type A/B is initialized with the trees given by figure 4.8. Then, positive and negative nodes are split in two according to the rules given by figures 4.9 and 4.10 until we reach atomic types. For both rules, splitting is performed by adding a new node

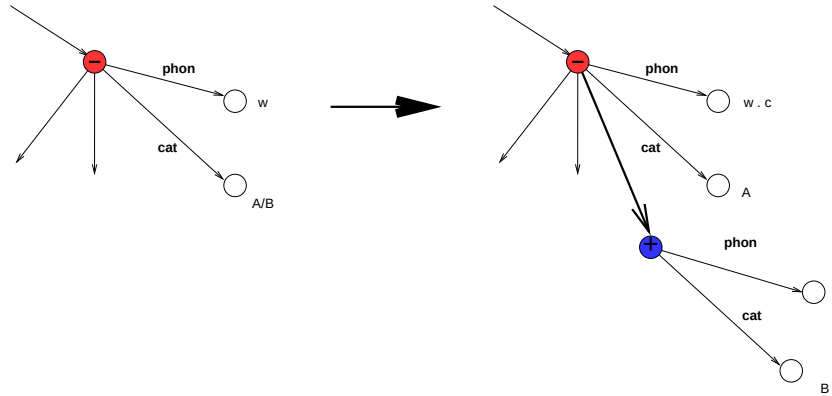
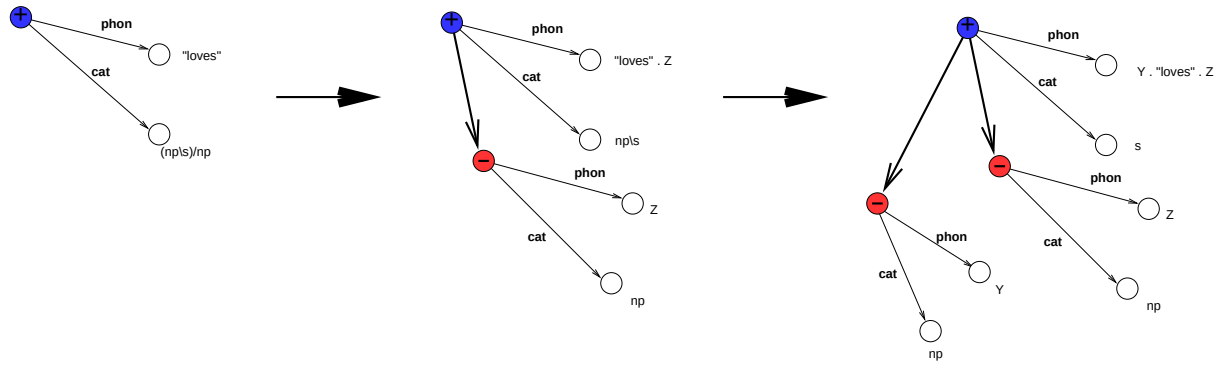


Figure 4.10: Splitting rule for negative nodes

⁶We only present the rewriting rules for the right division. The rules for the left division can be deduced easily.

Figure 4.11: Computation of $t^+(\text{"loves"} : (np \backslash s) / np)$

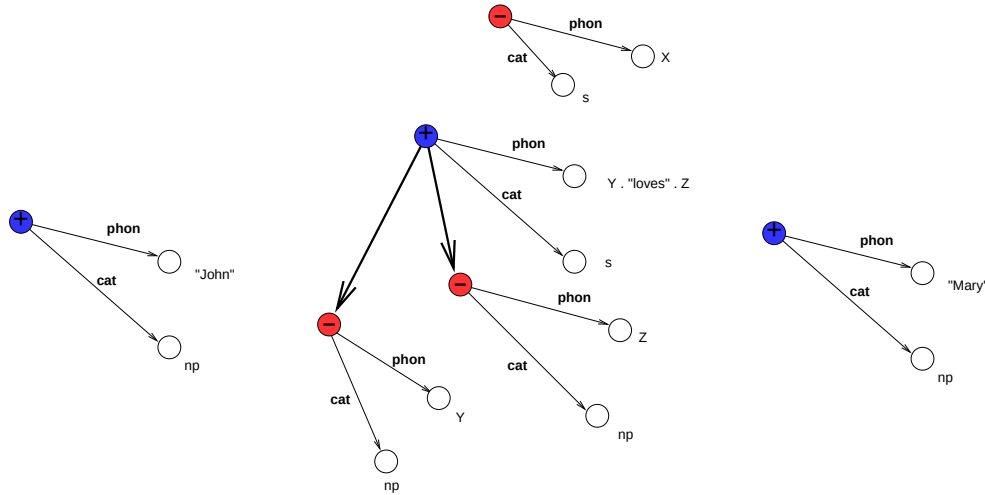
with an opposite polarity as a daughter of the initial node. This node is equipped with two features representing its phonological form and its syntactic category. The category and the phonological form of the mother node are changed by using a fresh constant c or a fresh variable X according to the polarity of the node.

Figure 4.11 shows an example of computation, that of $t^+(\text{"loves"} : (np \backslash s) / np)$. This example illustrates the fact that the functions t^+ and t^- identify the polarized trees computed for some syntactic types that differ on unessential points: for instance, they produce the same tree for the type $(np \backslash s) / np$ and $np \backslash (s / np)$.

After the definition of the translation of any typed expression from the framework of ABG to that of PUG, we deduce the translation of any AG grammar G into a FU grammar $t(G)$

Definition 4.3.1 For any AB grammar G , the corresponding FU grammar $t(G)$ is defined as follows:

- its lexicon associates to any word w the polarized trees $t^+(w : T)$ such that $T \in G(w)$;
- its axiom is $t^-(X : s)$ where X is a variable.

Figure 4.12: Initial frame provided by the grammar $t(G_0)$ for parsing *John loves Mary*

.The validity of the translation is established by the following theorem.

Theorem 4.3.1 For any AB grammar G , $L(G) = L(t(G))$.

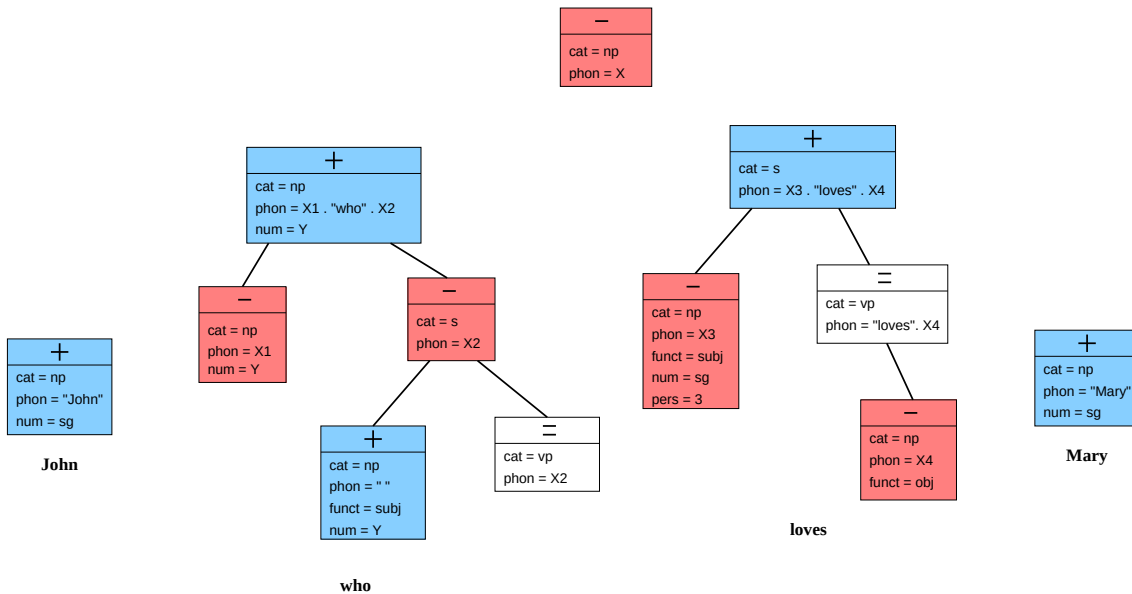


Figure 4.13: Lexical entries and axiom given by the PU grammar G_2 for parsing *John who loves Mary*

Exercise 4.3.1 *Prove theorem 4.3.1.*

For instance, let us translate the grammar G_0 into a PU grammar $t(G_0)$. Then let us use the resulting grammar for parsing the sentence *John loves Mary*. Figure 4.12 gives us the translation of the types of the sentence words and of the axiom. We recover the polarized trees presented in the introduction with a more simplified set of features. Then, the parsing process in the PUG framework mimics the parsing process in the ABG framework: every tree superposition is the translation of a forward or a backward application.

Exercise 4.3.2 *Extend the grammar G_0 to a grammar G_1 by designing syntactic types for the relative pronoun *who*, the personal pronouns *he* and *her*, the possessive determiner *her*, the common noun *brother* and the transitive verb *hates*. Then, parse the following sentences with G_1 and then with $t(G_1)$:*

He loves her passionately.

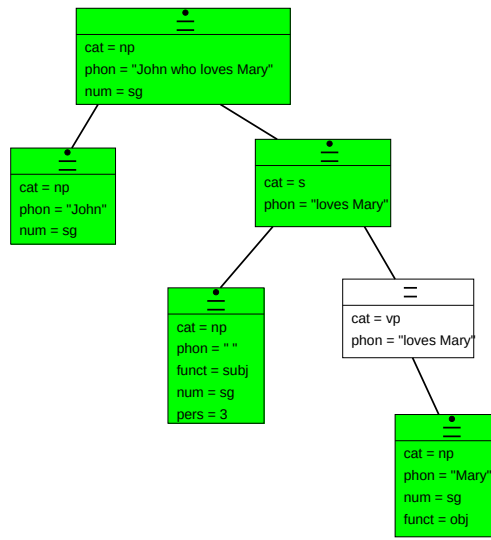
John who loves her hates her brother.

4.4 Expressivity and limitations of PUG

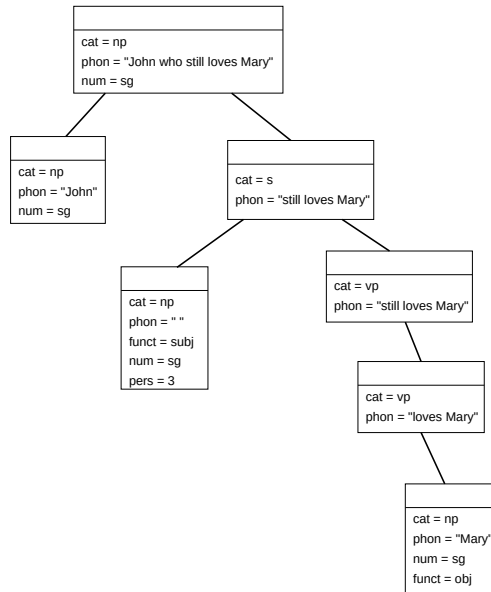
In the previous section, we have shown that AG grammars can be translated into PU grammars. The converse property is false because of the use of neutral nodes in PUG, which allows more flexible superpositions of trees.

Let us consider a polarized unification grammar G_2 which associates the polarized trees of figure 4.13 with the words of the expression *John who loves Mary* to parse. We add an axiom to these trees for expressing that the expected result of parsing is a noun phrase X . For making diagrams more readable, we have changed the graphical presentation of polarized trees: all features that are attached to a constituent are represented as a feature structure in a box, the colour of which depends on the polarity of the constituent.

After four superpositions, we obtain the neutral tree of figure 4.14. One of the superpositions concerns the trees associated with *who* and *loves*. In this superposition, three nodes of the first tree are merged with three nodes of the second tree and one merging affects two neutral nodes, which goes beyond what AB grammars are able to do.

Figure 4.14: Parse tree of *John who loves Mary*

Even if the expressiveness of PUG goes beyond ABG, it has strong limitations. An important limitation is that the formalism is unable to express underspecification. In particular, in every polarized tree, the position of every node with respect to the others is rigid. For instance, let us represent the syntax of the noun phrase *John who still loves Mary*. We obtain the tree given by figure 4.15. Then, let us imagine the polarized trees that can be associated to the words of the phrase by a PU grammar for building this tree. The adverb *still* is a modifier of the verb phrase *loves Mary* and it easy to design a corresponding tree with two nodes: a positive root with a negative daughter, both representing verb phrases. The lexical entry for the verb *loves* of figure 4.13 no longer holds because the sentence that has *loves* as head is now the grandmother of the verb phrase *loves Mary* and not its mother. We must change this lexical entry. It is easy to understand that the size of the lexicon will grow very quickly if we want to extend its coverage.

Figure 4.15: Parse tree of *John who still loves Mary*

Chapter 5

Underspecification and tree descriptions

5.1 Introduction

In the previous chapter, we have introduced PUG, a linguistic formalism that uses tree superposition guided by polarities as syntactic composition principle. The use of completely specified trees strongly limits the capability of the formalism to express linguistic generalizations.

For instance, consider the following noun phrases:

- (a) *Mary **whom** John **loves** passionately*
- (b) *Mary **whom** John claims to **love** passionately*
- (c) *Mary **whom** Peter tells John claims to **love** passionately*

The length of the object dependency from the transitive verb *love* to the relative pronoun *whom* increases when we go from expression (a) to expression (c), which needs three different lexical entries for *whom* in a PU grammar. Figure 5.1 shows three polarized trees which can represent such entries. In the three trees, the negative node N_3 of type *s* that represents the expected relative clause dominates the positive node N_4 of type *np* that represents the trace of the extracted object. The difference between the three trees lies in the length of the dominance chain, which increases from (a) to (c). We can imagine other examples in which the length is still greater, which expresses the fact that the corresponding dependency is unbounded. Every particular length requires a particular lexical entry in a PU grammar.

Such a multiplication of lexical entries is disastrous, especially if we envisage to build a wide coverage grammar. For avoiding this, it is crucial that the formalism be able to capture linguistic generalities. In the present case, it is to be hoped that we design only one lexical entry for the relative pronoun *whom* that captures all its uses. A way of doing this is to keep the dominance relation between N_3 and N_4 underspecified, as figure 5.2 shows. Since there is no limitation in the length of the chain that realizes the dominance relation between N_3 and N_4 , the underspecified tree represents an infinite number of completely specified PUG trees. Of course, since there are no constraints on the nature of the intermediate nodes that realize the dominance relation between N_3 and N_4 , some of these trees correspond to non-grammatical expressions. For the moment, we put this problem of overgeneration aside but we will bring it up again at the end of the chapter.

Now, we will show that there are some advantages in seeing the underspecified tree above from a new angle as a tree description. If we compare the trees expressing the syntactic behaviour of the relative pronouns *who* and *whom* given by the respective figures 4.13 and 5.2, we remark that they share a common part. A way of expressing this is to characterize every tree, underspecified or not, by a set of elementary constraints on their nodes, representing constituents, and features values, representing morpho-syntactic properties of these constituents. This set is constituted of three parts:

- the first part gathers constraints on polarities of nodes;

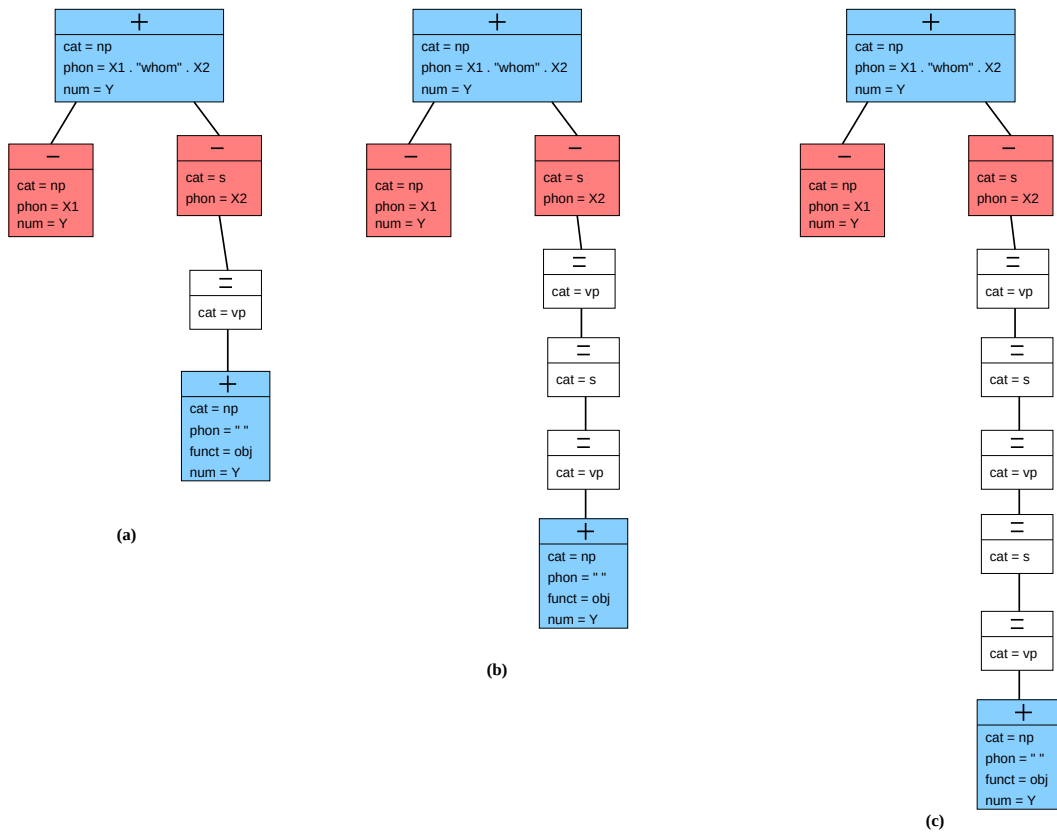


Figure 5.1: Lexical entries for the relative pronoun *whom* in a PU grammar

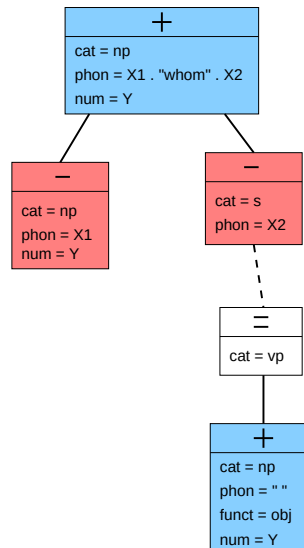


Figure 5.2: Underspecified tree factorizing several lexical entries for the relative pronoun *whom* from a PU grammar

- the second part gathers constraints expressing relations between nodes and feature values or relations between only feature values;
- the third part gathers constraints expressing dominance relations between nodes.

Such a set of constraints does not necessarily describe a unique tree but rather a family of trees and it is called a *tree description*. In this way, the two trees corresponding to the relative pronouns *who* and *whom* can be characterized by two tree descriptions D_{who} and D_{whom} . For stating them, we need the following relations:

- $pol(N, p)$, which means that node N has polarity p ;
- $N : f = V$, which means that node N has feature f with value V ;
- $N > M$, which means that node N immediately dominates node M ;
- $N \overset{*}{>} M$, which means that node N dominates node M in a large sense, that is they can be equal.

With these relations, D_{who} and D_{whom} are defined as follows:

$$\begin{aligned}
 D_{who} = \{ & pol(N_1, +), (N_1 : cat = np), (N_1 : phon = X_1, "who".X_2), (N_1 : num = Y), \\
 & pol(N_2, -), (N_2 : cat = np), (N_2 : phon = X_1), (N_2 : num = Y), \\
 & pol(N_3, -), (N_3 : cat = s), (N_2 : phon = X_2), \\
 & pol(N_4, +), (N_4 : cat = np), (N_4 : phon = " "), (N_4 : funct = subj), (N_4 : num = Y), \\
 & pol(N_5, =), (N_5 : cat = vp), (N_5 : phon = X_2), \\
 & N_1 > N_2, N_1 > N_3, N_3 > N_4, N_3 > N_5 \\
 & \} \\
 D_{whom} = \{ & pol(N_1, +), (N_1 : cat = np), (N_1 : phon = X_1, "who".X_2), (N_1 : num = Y), \\
 & pol(N_2, -), (N_2 : cat = np), (N_2 : phon = X_1), (N_2 : num = Y), \\
 & pol(N_3, -), (N_3 : cat = s), (N_2 : phon = X_2), \\
 & pol(N_4, +), (N_4 : cat = np), (N_4 : phon = " "), (N_4 : funct = obj), (N_4 : num = Y), \\
 & pol(N_5, =), (N_5 : cat = vp) \\
 & N_1 > N_2, N_1 > N_3, N_3 \overset{*}{>} N_5, N_5 > N_4 \\
 & \}
 \end{aligned}$$

Figure 5.2 is a graphical representation of the description D_{whom} in the shape of an underspecified tree, whereas the three trees on figure 5.1 are three realizations of this description, three models in other terms. The common part of the trees corresponding to the relative pronouns *who* and *whom* appears as the intersection D_{rel} of the descriptions D_{who} and D_{whom} , that is:

$$\begin{aligned}
 D_{rel} = \{ & pol(N_1, +), (N_1 : cat = np), (N_1 : phon = X_1, "who".X_2), (N_1 : num = Y), \\
 & pol(N_2, -), (N_2 : cat = np), (N_2 : phon = X_1), (N_2 : num = Y), \\
 & pol(N_3, -), (N_3 : cat = s), (N_2 : phon = X_2), \\
 & pol(N_4, +), (N_4 : cat = np), (N_4 : phon = " "), (N_4 : num = Y), \\
 & pol(N_5, =), (N_5 : cat = vp) \\
 & N_1 > N_2, N_1 > N_3 \\
 & \}
 \end{aligned}$$

With a more refined notion of intersection between descriptions, we can consider that the relation $N_3 \overset{*}{>} N_5$ belongs to the intersection D_{rel} because $N_3 > N_5$ entails $N_3 \overset{*}{>} N_5$.

The flexibility of the notion of tree description, which allows to combine elementary properties freely, makes it well-suited to the hierarchical organization of grammars according to the natural relation of inheritance between tree descriptions represented by inclusion. For instance, since $D_{rel} \subset D_{who}$ and $D_{rel} \subset D_{whom}$, we deduce that D_{who} and D_{whom} inherit from the common description D_{rel} .

5.2 Polarized Tree Description Grammars

The notion of tree description was first introduced by M. Marcus, D. Hindle and M. Fleck for representing syntactic ambiguity in natural languages [MHF83]. A few years later, it was taken again by K. Vijay-Shanker for representing the formalism of Tree Adjoining Grammars in a monotone way [VS92]. Then,

it was formalized in a logical framework by J. Rogers and K. Vijay-Shanker [RVS94]. Recently, new grammatical formalisms have been developed around the notion of tree description [Kal99, RVSW01].

Here, we will use the notion of tree description for designing a grammatical formalism that amalgamates it with the notion of polarity. We call this formalism Polarized Tree Description Grammars and we will refer to it by the acronym PTDG. In a similar manner as for other formalisms, we will use the term PTD grammar to refer to a particular Polarized Tree Description grammar.

5.2.1 The form of polarized tree descriptions

In order to define the form of polarized tree descriptions, we assume a signature constituted of three parts:

- a set *Nodes* of variables representing nodes of tree descriptions;
- a set *Feat* of features identical to that used for defining PUG; every element f of *Feat* is associated with a domain D_f of constant values;
- a set *Val* of variables representing unspecified feature values.

Then, a polarized tree description is defined as follows:

Definition 5.2.1 *A polarized tree description is a finite set of relations in one of the following form:*

- $pol(N, P)$ with $N \in Nodes$ and $P \in \{+, -, =, \doteq\}$;
- $(N : F = V)$ with $N \in Nodes$, $F \in Feat$ and $V \in D_F \cup Val$, except if $F = phon$: in this case, V is a concatenation of elements from $D_{phon} \cup Val$;
- $N > M$ with $N, M \in Nodes$;
- $N \overset{*}{>} M$ with $N, M \in Nodes$.

Instead of a set presentation, we can choose a logical presentation of tree descriptions in the form of conjunctions of predicates [RVS94]. Such a presentation allows two kinds of combination between descriptions: conjunction and disjunction. Since we only use conjunction, the set presentation is sufficient for us, and, in this presentation, conjunction is expressed by set union. Some descriptions play a special role, that are *saturated descriptions*, which include neither positive nor negative nodes.

5.2.2 The models of tree descriptions

A tree description is a way of representing a set of trees in a compact way, each tree being a model of this description. This relationship between descriptions and models can be formalized in a very strict manner in a logical framework with model theory but we will not go to such a degree of formalization. For defining the models of a tree description, we have first to define the form of such models, that are syntactic trees.

Definition 5.2.2 *A syntactic tree is a finite tree, the nodes of which are labelled with feature structures, each feature f having a constant coming from D_f as value.*

The form of models being fixed, we must now establish their relationship with descriptions.

Definition 5.2.3 *A model of a polarized tree description D is a triple (T, I, Val) in which T is a syntactic tree, I is an interpretation function which maps every node variable of D to a node of T and Val is a valuation function which maps every feature variable of D to a feature constant value.*

The functions I and V must verify the following conditions:

- if $(N : F = V) \in D$, then $(F = Val(V))^1$ belongs to the feature structure labelling $I(N)$;
- if $(N > M) \in D$, then $I(M)$ is a daughter of $I(N)$ in T ;
- if $(N \overset{*}{>} M) \in D$, then $I(M)$ is a descendant of $I(N)$ or equal to $I(N)$.

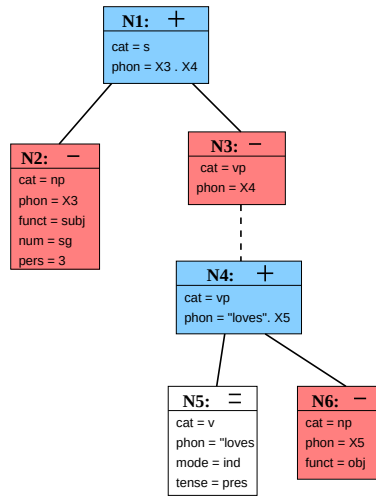


Figure 5.3: Polarized tree description representing the SVO construction headed by the transitive verb *loves*

Figure 5.3 shows an example of tree description, D_{loves} , representing the canonical SVO construction headed by the transitive verb *loves*. This graphical representation is the exact translation of the following enumerative presentation of D_{loves} :

$$\begin{aligned}
 D_{loves} = \{ & \text{pol}(N_1, +), (N_1 : \text{cat} = s), (N_1 : \text{phon} = X_3, \text{"loves"}.X_4) \\
 & \text{pol}(N_2, -), (N_2 : \text{cat} = np), (N_2 : \text{phon} = X_3), (N_2 : \text{funct} = \text{subj}), \\
 & (N_2 : \text{num} = sg), (N_2 : \text{pers} = 3), \\
 & \text{pol}(N_3, -), (N_3 : \text{cat} = vp), (N_3 : \text{phon} = X_4), \\
 & \text{pol}(N_4, +), (N_4 : \text{cat} = vp), (N_4 : \text{phon} = \text{"loves"}.X_5), \\
 & \text{pol}(N_5, =), (N_5 : \text{cat} = v), (N_5 : \text{phon} = \text{"loves"}), (N_5 : \text{mode} = ind), (N_5 : \text{tense} = pres), \\
 & \text{pol}(N_6, -), (N_6 : \text{cat} = np), (N_6 : \text{phon} = X_5), (N_6 : \text{funct} = obj) \\
 & N_1 > N_2, N_1 > N_3, N_3 \overset{*}{>} N_4, N_4 > N_5, N_4 > N_6 \\
 & \}
 \end{aligned}$$

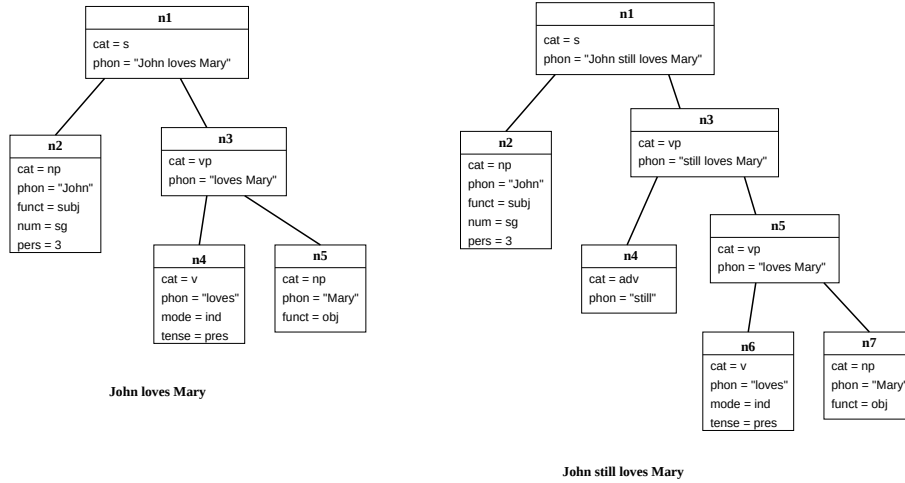


Figure 5.4: Syntactic trees models of the tree description D_{loves}

¹ V can be a variable or a constant, or a concatenation of variables and constants in the case of the feature *phon*; for this, the function *Val* is extended to all feature values in a natural way.

Figure 5.4 presents two different models of the description D_{loves} , which respectively represent the syntax of the sentences *John loves Mary* and *John still loves Mary*. The syntactic tree T_1 of the sentence *John loves Mary*, with the following interpretation I_1 and valuation Val_1 functions, is a model of the description D_{loves} :

$$I_1 = \{N_1 \rightarrow n_1, N_2 \rightarrow n_2, N_3 \rightarrow n_3, N_4 \rightarrow n_3, N_5 \rightarrow n_4, N_6 \rightarrow n_5\}$$

$$Val_1 = \{X_3 \rightarrow \text{"John"}, X_4 \rightarrow \text{"loves Mary"}, X_5 \rightarrow \text{"Mary"}\}$$

The syntactic tree T_2 of the sentence *John still loves Mary*, with the following interpretation I_2 and valuation Val_2 functions, is a model of the description D_{loves} :

$$I_2 = \{N_1 \rightarrow n_1, N_2 \rightarrow n_2, N_3 \rightarrow n_3, N_4 \rightarrow n_5, N_5 \rightarrow n_6, N_6 \rightarrow n_7\}$$

$$Val_2 = \{X_3 \rightarrow \text{"John"}, X_4 \rightarrow \text{"still loves Mary"}, X_5 \rightarrow \text{"Mary"}\}$$

5.2.3 Minimal and neutral models of tree descriptions

When a tree description expresses the specification of a whole sentence, the general notion of model of a description is not sufficient for two reasons:

1. a description has an infinite number of models and it is necessary to exhibit privileged models representing the syntax of the sentence; such models have a property of *minimality*: informally, a model is minimal when it adds a minimum of information with respect to the description;
2. polarities are not taken into account in the definition of models; however, the models representing the syntax of a sentence must be saturated, which is expressed with a property of *neutrality*.

Let us define these two properties of models more precisely.

Definition 5.2.4 A model (T, I, Val) of a description D is minimal if every immediate dominance relation in T interprets an immediate dominance relation in D .

In our last example, (T_1, I_1, Val_1) is a minimal model of D_{love} . At the opposite, (T_2, I_2, Val_2) is not a minimal model because the immediate dominance relation between n_3 and n_5 interprets no immediate dominance relation from D_{love} .

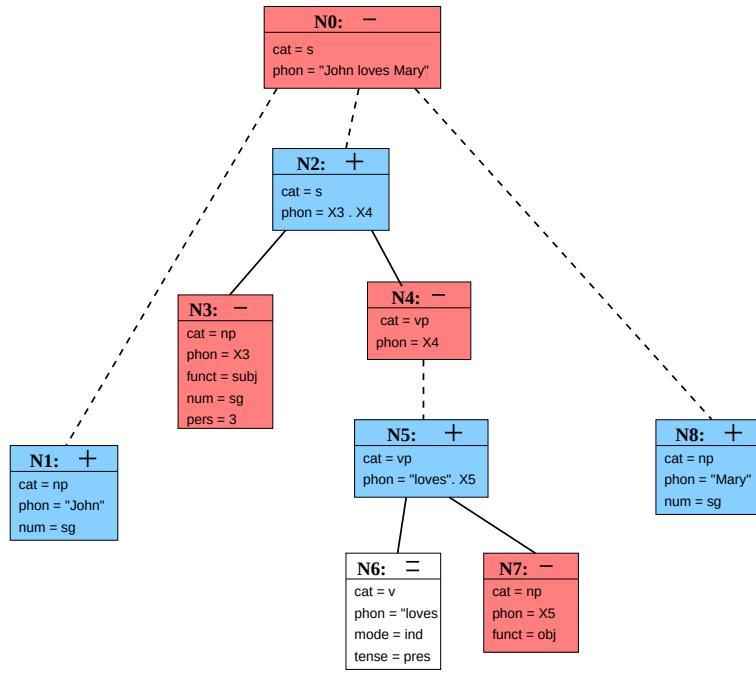
Definition 5.2.5 A model (T, I, Val) of a description D is neutral if for every node n of T , the set $I^{-1}(n)$ has one of the following properties:

- for every relation $pol(N, P) \in D$ such that $N \in I^{-1}(n)$, P is the neutral polarity $=$;
- for every relation $pol(N, P) \in D$ such that $N \in I^{-1}(n)$, P is the neutral polarity $=$, except one case where P is the saturated polarity $\circ$;
- for every relation $pol(N, P) \in D$ such that $N \in I^{-1}(n)$, P is the neutral polarity $=$, except two cases where P is the positive and negative polarities $+$ and $-$.

According to this definition, the nodes N of a description D that are not polarized, that is for which there exists no relation $pol(N, P) \in D$, behave like neutral nodes, in other words if there exists a relation $pol(N, =) \in D$. Therefore, we can consider that all nodes of any description are polarized.

In our last example, (T_1, I_1, Val_1) and (T_2, I_2, Val_2) are not neutral models of D_{love} : in T_2 , the only node that respects the neutrality condition is n_4 whereas in T_1 two nodes verify this condition: n_3 and n_4 . Such results are expected because the grammatical sentences *John loves Mary* and *John still loves Mary* cannot be built from the unique description associated with *loves*.

A model that is minimal and neutral at the same time is said to be *grammatical* and a description that has a grammatical model, is said to be grammatical too. For instance, consider the description D_0 from figure 5.5. It is the union of the descriptions associated by the lexicon defining a PDT grammar to the words of the sentence *John loves Mary* with an additional root node representing an axiom, which expresses that the expected result of parsing is the sentence *John loves Mary*. The enumerative presentation of D_0 is the following:

Figure 5.5: Description D_0 associated with the sentence *John loves Mary*

$$\begin{aligned}
 D_0 = \{ & \text{pol}(N_0, -), (N_0 : \text{cat} = s), (N_0 : \text{phon} = \text{"John loves Mary"}), \\
 & \text{pol}(N_1, +), (N_1 : \text{cat} = np), (N_1 : \text{phon} = \text{"John"}), (\text{num} = sg), \\
 & \text{pol}(N_2, +), (N_2 : \text{cat} = s), (N_2 : \text{phon} = X_3, \text{"loves"}.X_4), \\
 & \text{pol}(N_3, -), (N_3 : \text{cat} = np), (N_3 : \text{phon} = X_3), (N_3 : \text{funct} = \text{subj}), \\
 & (N_3 : \text{num} = sg), (N_3 : \text{pers} = 3), \\
 & \text{pol}(N_4, -), (N_4 : \text{cat} = vp), (N_4 : \text{phon} = X_4), \\
 & \text{pol}(N_5, +), (N_5 : \text{cat} = vp), (N_5 : \text{phon} = \text{"loves"}.X_5), \\
 & \text{pol}(N_6, =), (N_6 : \text{cat} = v), (N_6 : \text{phon} = \text{"loves"}), (N_6 : \text{mode} = \text{ind}), (N_6 : \text{tense} = \text{pres}), \\
 & \text{pol}(N_7, -), (N_7 : \text{cat} = np), (N_7 : \text{phon} = X_5), (N_7 : \text{funct} = \text{obj}), \\
 & \text{pol}(N_8, +), (N_8 : \text{cat} = np), (N_8 : \text{phon} = \text{"Mary"}), (\text{num} = sg) \\
 & N_0 \overset{*}{>} N_1, N_0 \overset{*}{>} N_2, N_0 \overset{*}{>} N_8, N_2 > N_3, N_2 > N_4, \\
 & N_4 \overset{*}{>} N_5, N_5 > N_6, N_5 > N_7 \\
 & \}
 \end{aligned}$$

Consider the triple (T_1, I'_1, Val'_1) such that T_1 is the syntactic tree of the sentence *John loves Mary* given by figure 5.4 and the interpretation and valuation functions I'_1 and Val'_1 are the following:

$$\begin{aligned}
 I'_1 &= \{N_0 \rightarrow n_1, N_1 \rightarrow n_2, N_2 \rightarrow n_1, N_3 \rightarrow n_2, N_4 \rightarrow n_3, N_5 \rightarrow n_3, N_6 \rightarrow n_4, N_7 \rightarrow n_5, N_8 \rightarrow n_5\} \\
 Val'_1 &= \{X_3 \rightarrow \text{"John"}, X_4 \rightarrow \text{"loves Mary"}, X_5 \rightarrow \text{"Mary"}\}
 \end{aligned}$$

We can verify that (T_1, I'_1, Val'_1) is a minimal and neutral model of D_0 ; therefore, D_0 and the sentence *John loves Mary* are grammatical.

Of course, the grammaticality of a sentence is relative to a particular PDT grammar since the descriptions associated with its words are provided by a lexicon and if we consider only lexicalized grammars, this lexicon defines the grammar totally; we must only add special descriptions, the axioms of the grammar, which represent the expected results of parsing. For any word w of the language, we refer to the set of descriptions associated with the word w by the lexicon of the grammar G by $G(w)$. Then, the axioms of a grammar are associated with all expressions $expr$ of the language in the shape of descriptions denoted $Ax(expr)$ defined as follows: $Ax(expr) = \{\text{pol}(N_0, -), (N_0 : \text{cat} = s), (N_0 : \text{phon} = expr)\}$. Finally, the language generated by a PDT grammar is defined as follows:

Definition 5.2.6 *The language $L(G)$ generated by a PDT grammar G is the set of sentence $w_1 \dots w_n$*

that verify the following condition: there exists a sequence $D_1, \dots, D_n$ of polarized tree descriptions such that for $0 \leq i \leq n$, $D_i \in G(w_i)$ and there is a minimal and neutral model of the description $D_1 \uplus \dots \uplus D_n \uplus Ax(w_1 \dots w_n)$.

5.2.4 Computation of grammatical models by neutralizing polarities

The definition of grammatical models of polarized tree descriptions allows to verify if a given syntactic tree is a grammatical model of a given polarized tree description but it does not provide any way for computing such a model from a given description.

For this, we use an operation, called *polarity neutralization*, which consists in merging a positive node N_1 with a negative node N_2 of a description D and in replacing their polarities by the polarity $\dot{=}$. If we denote the resulting description by D' , we write: $D \xrightarrow{N_1=N_2} D'$.

This operation can be viewed as a way for specifying a description: neutralizing two nodes N_1 and N_2 in a description D amounts to selecting the grammatical models of D that identify N_1 and N_2 with their interpretation function. More precisely, if $D \xrightarrow{N_1=N_2} D'$ and if (T, I, Val) is a grammatical model of D , such that $I(N_1) = I(N_2)$, then we obtain a model of D' by replacing the interpretation I of D with $I[N_1 = N_2]$, which interprets every node of D' present in D in the same way as in D and the node replacing N_1 and N_2 with their common value $I(N_1) = I(N_2)$.

In the iteration of the operation of neutralization, two cases happen:

- any polarity neutralization is no longer possible and we can conclude that the description is not grammatical;
- the process ends with a saturated description and we can conclude that every grammatical model of this description corresponds to a grammatical model of the initial description.

In order to formalize the justification of the operation of neutralization, we need to define the notion of *description neutralization*.

Definition 5.2.7 A neutralization of a description D_0 is a sequence $D_0 \xrightarrow{N_1^0=N_2^0} D_1 \xrightarrow{N_1^1=N_2^1} \dots \xrightarrow{N_1^p=N_2^p} D_p$, possibly empty, such that D_p is saturated.

With this notion of description neutralization, we can establish the following theorem which grounds the operation of polarity neutralization.

Theorem 5.2.1 For any description D_p that ends a neutralization of a description D_0 and for any grammatical model (T, I_p, Val) of D_p , there exists a grammatical model (T, I_0, Val) of D_0 such that: $I_p = I_0[N_1^0 = N_2^0][N_1^1 = N_2^1] \dots [N_1^p = N_2^p]$.

Conversely, for any grammatical model (T, I_0, Val) of D_0 , there exists a neutralization of D_0 $D_0 \xrightarrow{N_1^0=N_2^0} D_1 \xrightarrow{N_1^1=N_2^1} \dots \xrightarrow{N_1^p=N_2^p} D_p$ such that $(T, I_0[N_1^0 = N_2^0][N_1^1 = N_2^1] \dots [N_1^p = N_2^p], Val)$ is a grammatical model of D_p .

A corollary of the second part of this theorem is that a description for which there exists no neutralization is not grammatical.

Exercise 5.2.1 Prove theorem 5.2.1.

Such a theorem guarantees the correctness and the completeness of the method that consists in iterating polarity neutralizations until we obtain a saturated description for computing grammatical models of a given description. Of course, the resulting description is not necessarily completely specified and may still include underspecified dominance relations between nodes but we can hope that, in realistic applications residual underspecification will be not very important.

Take again the description D_0 from figure 5.5 and consider the following neutralization of D_0 : $D_0 \xrightarrow{N_1=N_3} D_1 \xrightarrow{N_7=N_8} D_2 \xrightarrow{N_4=N_5} D_3 \xrightarrow{N_0=N_1} D_4$. The resulting description D_4 is saturated and completely specified: it has exactly one grammatical model. This model is directly deduced from the model (T_1, I'_1, Val'_1) previously defined for D_0 . Finally, it is easy to show that D_0 has exactly one grammatical model, modulo possible renaming.

“John”: np
“Mary”: np
“loves”: $(np \backslash s) / np$
“whom”: $(np \backslash np) / (s / np)$

Figure 5.6: The toy Lambek grammar G_2

5.3 The example of Lambek Grammars

5.3.1 The logical presentation of Lambek Grammars

We have seen in section 4.4 that PUG presents a strong limitation because it cannot express underspecification. ABG, which is a particular instantiation of PUG, inherits these limitations. For going beyond them, the rewriting system of ABG was enriched with a collection of new rules expressing composition, type raising ... [Moo88]

The Lambek calculus, designed by J. Lambek in the 50s [Lam58], unify all these rules, which can look ad hoc, into a logical system: syntactic types become logical formulas and syntactic composition rules become logical inference rules, so that parsing takes the form of logical deduction.

In linguistic applications, we usually resort to the implicative part of the Lambek Calculus, which includes two logical connectives: backward implication ($\backslash$) and forward implication ($/$), which correspond to the backward and the forward divisions of ABG. In the following, we will refer to this system by LG (for Lambek Grammars). The two ABG rules become the two LG elimination rules and they are completed by two corresponding introduction rules for building a completely symmetrical logical system:

$$\begin{array}{c}
 \frac{w_1 : B \quad w_2 : B \backslash A}{w_1 . w_2 : A} (\backslash_E) \qquad \frac{w_1 : A / B \quad w_2 : B}{w_1 . w_2 : A} (/_E) \\
 \\
 \frac{\begin{array}{c} [x : B] \\ \vdots \\ x . w : A \end{array}}{w : B \backslash A} (\backslash_I) \qquad \frac{\begin{array}{c} [x : B] \\ \vdots \\ w . x : A \end{array}}{w : A / B} (/_I)
 \end{array}$$

In a similar way as natural deduction for intuitionistic logic, the rules above define the natural deduction system of LG. In the introduction rules, $[x : B]$ represents a hypothesis that is cancelled in the inference of the conclusion $w : B \backslash A$ or $w : A / B$. Such a mechanism allows to perform hypothetical reasoning: we add new hypotheses in a proof when it is necessary and then, we cancel them in the course of the proof.

By combining applications of these rules, we build derivation trees, which are realizations of derivations. For denoting derivations, we take again the notation of ABG: if a derivation tree has $w_1 . w_2 \dots w_n : C$ as its root with the sequence $w_1 : H_1, w_2 : H_2, \dots, w_n : H_n$ as its leaves, we denote the corresponding derivation $w_1 : H_1, w_2 : H_2, \dots, w_n : H_n \xrightarrow{*} w_1 . w_2 \dots w_n : C$.

In the same way as in ABG, we define Lambek grammars by their lexicon, which links every word of the language with a set of syntactic types. The language generated by a particular Lambek grammar is defined in the same way as for an AB grammar.

As an example, consider the small toy grammar G_2 given by figure 5.6. The derivation tree given by figure 5.7 justifies the derivation *“Mary”* : np , *“whom”* : $(np \backslash np) / (s / np)$, *“John”* : np , *“loves”* : $(np \backslash s) / np \xrightarrow{*} \text{“Mary whom John loves”} : np$.

5.3.2 Translation in the PTDG formalism

Again, this translation takes the rewriting system designed for translating AGG in PUG but using polarized tree descriptions instead of polarized trees. Instead of the functions t^+ and t^- , we consider

$$\begin{array}{c}
\frac{\frac{\frac{\text{"loves"}: (np \backslash s) / np \quad [x: np]}{(\text{"loves"} \cdot x: np \backslash s)} (/E)}{\text{"John"}: np \quad \frac{\text{"loves"}: (np \backslash s) / np \quad [x: np]}{(\text{"loves"} \cdot x: np \backslash s)} (/E)} (\backslash_E) \\
\frac{\text{"whom"}: (np \backslash np) / (s / np) \quad \frac{\text{"John loves"}: s}{\text{"John loves"}: s / np} (/I)}{\text{"Mary"}: np \quad \frac{\text{"John loves"}: s / np}{\text{"John loves"}: s / np} (\backslash_E)} (\backslash_E) \\
\frac{\text{"Mary"}: np \quad \text{"whom John loves Mary"}: np \backslash np}{\text{"Mary whom John loves"}: np} (\backslash_E)
\end{array}$$

Figure 5.7: Derivation of the noun phrase *Mary whom John loves* with the grammar G_2



Figure 5.8: Description initialization for the computation of $d^-(w : A/B)$ and $d^+(w : A/B)$

two functions d^+ and d^- , which translate any expression w with the syntactic type A from LG into two polarized tree descriptions $d^+(w : A)$ and $d^-(w : A)$ of PTDG.

The computation of $d^-(w : A/B)$ and $d^+(w : A/B)$ for any expression w of the type A/B is initialized with the tree descriptions given by figure 5.8². Then, positive and negative nodes are split in two according to the rules given by figures 5.9 and 5.10 until we reach atomic types. The difference with

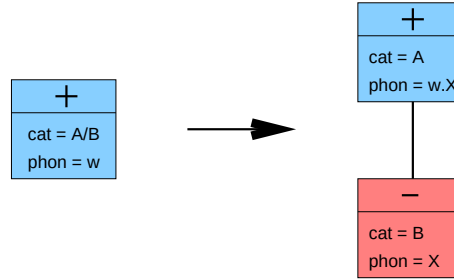


Figure 5.9: Splitting rule for positive nodes

respect to the rules for the translation of ABG in PUG, is that the relation between the generated nodes in the case of splitting a negative node is a dominance relation. Hence, the formalism allows to express underspecification.

Figure 5.11 shows an example of computation, that of $d^+(\text{"whom"} : (np \backslash np) / (s / np))$.

After the definition of the translation of any typed expression from the framework of LG to that of PTDG, we deduce the translation of any Lambek grammar G into a PTD grammar $d(G)$

Definition 5.3.1 *For any Lambek grammar G , the corresponding PTD grammar $d(G)$ is defined as follows:*

- *its lexicon associates every word w to the set $d(G)(w)$ of polarized tree descriptions $d^+(w : T)$ such that $T \in G(w)$;*
- *its axioms are all descriptions $d^-(\text{expr} : s)$ where expr is any concatenation of words.*

.The validity of the translation is established by the following theorem.

²We only present the rewriting rules for the forward implication. The rules for the backward implication can be deduced easily.

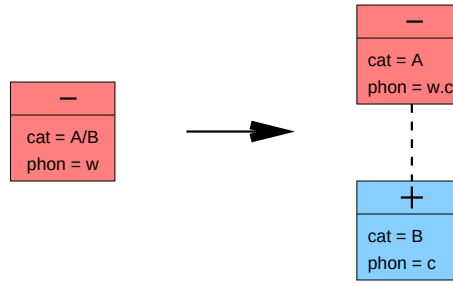
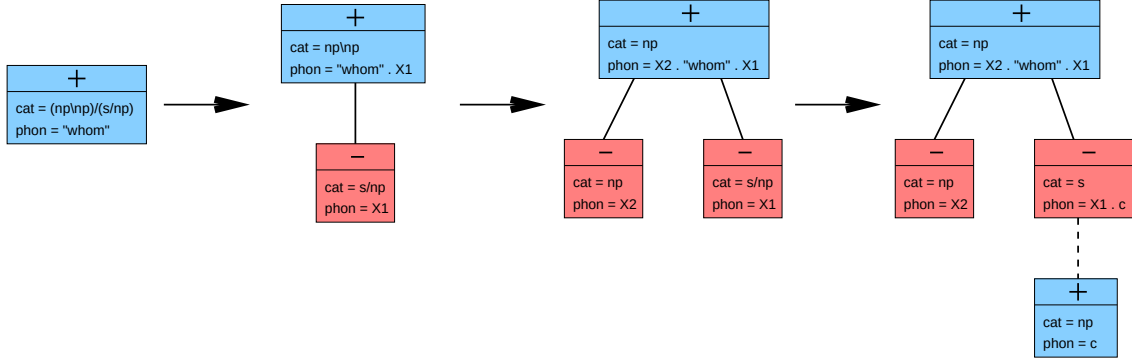


Figure 5.10: Splitting rule for negative nodes

Figure 5.11: Computation of $d^+(\text{"whom"} : (np \backslash np) / (s \backslash np))$

Theorem 5.3.1 For any Lambek grammar G , $L(G) = L(d(G))$.

Exercise 5.3.1 Prove theorem 5.3.1.

For instance, let us translate the grammar G_2 into a PTD grammar $d(G_2)$. Then let us use the resulting grammar for parsing the noun phrase *Mary whom John loves*. Figure 5.12 gives us the translation of the types of the words building the expression. For expressing the expected result, we must add an axiom, which says that this result is a noun phrase. Then, the parsing process in the PTDG framework mimics the parsing process in the LG framework.

Exercise 5.3.2 Extend the grammar G_2 to a grammar G_3 by designing syntactic types for the relative pronoun *who*, the personal pronouns *he*, *him* and *her*, the possessive determiners *his* and *her*, the common noun *brother*, the adverb *passionately* as a modifier of a verb phrase, the transitive verb *hates* and the proper noun *Peter*. Then, parse the following sentences with G_3 and then with $d(G_3)$:

- He loves her.
- John who loves her hates her brother.
- Mary whom he loves hates Peter.
- Mary whom John who hates Peter loves hates his brother.

Try to parse *Mary whom John who loves hates Peter hates him* and *Mary whom John loves passionately*.

5.4 Possible enrichment of the PTDG formalism

In the PTDG formalism, word order is expressed in phonological features but it could be also expressed by *precedence relations* between nodes. The advantage is that it makes possible to distinguish immediate precedence relations from underspecified precedence relations. In the following, we write $N_1 \prec N_2$ for

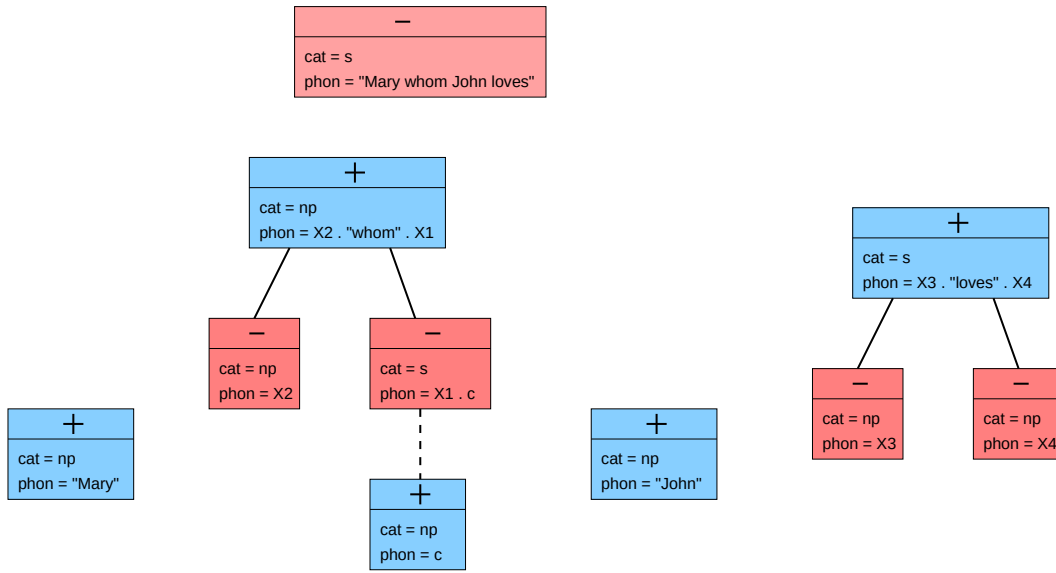


Figure 5.12: Initial frame provided by the grammar $d(G_2)$ for parsing *Mary whom John loves*

saying that node N_1 precedes node N_2 immediately and we write $N_1 \overset{+}{\prec} N_2$ for saying that node N_1 precedes node N_2 strictly.

We can also add new constraints, which are useful for increasing computational efficiency or the expressivity of the formalism. For instance, $N > \{N_1, \dots, N_p\}$ means that node N has exactly p different daughter nodes $N_1, \dots, N_p$. Such a constraint is very useful for increasing efficiency in the search for models of descriptions.

Concerning the expressivity of the formalism, underspecified dominance relations are too lax and entail overgeneration, especially because they do not take islands constraints in long distance dependencies. In order to avoid this problem we can use constrained dominance relations in the following form: $N_1 \overset{*}{>} \{f_1 = v_1, \dots, f_n = v_n\} N_2$, where $\{f_1 = v_1, \dots, f_n = v_n\}$ is feature structure which must unify with that of every node N such that $N_1 \overset{*}{>} N$ and $N \overset{*}{>} N_2$.

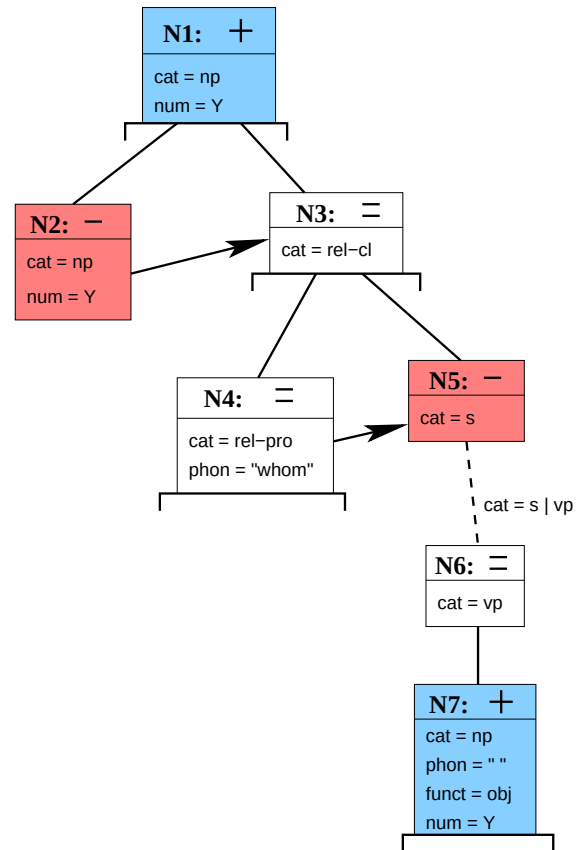
A last enrichment consists in allowing disjunctions in feature values. We write $f = v_1 | v_2 | \dots | v_n$ for saying that the feature f has one of the following values $v_1, v_2, \dots, v_n$.

In order to illustrate these extensions, again let us take the polarized description associated with the relative pronoun *whom* of figure 5.1 and let us enrich it with the constraints presented above. We obtain the following description D'_{whom} , the graphical presentation of which is given by figure 5.13.

$$D_{whom2} = \{ \begin{array}{l} pol(N_1, +), (N_1 : cat = np), (N_1 : num = Y), \\ pol(N_2, -), (N_2 : cat = np), (N_2 : num = Y), \\ pol(N_3, =), (N_3 : cat = relcl), \\ pol(N_4, =), (N_4 : cat = relpro), \\ pol(N_5, -), (N_5 : cat = s), \\ pol(N_6, =), (N_6 : cat = vp), \\ pol(N_7, +), (N_7 : cat = np), (N_7 : phon = " "), (N_7 : funct = obj), (N_7 : num = Y), \\ N_1 > \{N_2, N_3\}, N_3 > \{N_4, N_5\}, N_4 > \{ \}, N_5 \overset{*}{>} \{cat = s | vp\} N_6, N_6 > N_7, N_7 > \{ \} \end{array} \}$$

On figure 5.13, immediate precedence relations between nodes are represented by arrows and arity constraints in the form of $N > \{N_1, \dots, N_p\}$ are represented by square brackets at the bottom of boxes representing nodes.

Exercice 5.4.1 Design a grammar G_4 in an extension of the PTDG formalism which accepts the three sentences at the beginning of the chapter and which rejects the sentence *Mary whom John who loves hates Peter hates him*.

Figure 5.13: Enrichment of a lexical entry for the relative pronoun *whom*

Bibliography

- [Adj35] K. Adjukiewicz. Die syntaktische konnexität. *Studia Philosophica*, 1:1–27, 1935.
- [BH53] Y. Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29:47–58, 1953.
- [Bla00] P. Blackburn. Representation, reasoning, and relational structures: a hybrid logic manifesto. *Logic Journal of the IGPL*, 8(3):339–625, 2000.
- [DT99] D. Duchier and S. Thater. Parsing with tree descriptions: a constraint based approach. In *Natural Language Understanding and Logic Programming NLULP’99, Dec 1999, Las Cruces, New Mexico*, 1999.
- [Dym97] Marc Dymetman. Interaction-free grammars, chart parsing, and the compact representation of ambiguity. In *Proc of IJCAI*, Nagoya, 1997.
- [Ear70] S.W. Earley. An efficient context-free parsing algorithm. *Communication ACM*, 13(2), 1970.
- [Kah04] S. Kahane. Grammaires d’unification polarisées. In *TALN2004, Fès, Maroc*, 2004. Submitted.
- [Kal99] L. Kallmeyer. *Tree Description Grammars and Underspecified Representations*. PhD thesis, Universität Tübingen, 1999.
- [Kas65] T. Kasami. An efficient recognition and syntax analysis algorithm for context-free languages. Technical Report R-257, University of Hawaii, 1965.
- [KR86] R. Kasper and W. Rounds. A logical semantics for feature structures. In *Proceedings of the 24th Annual Meeting of the ACL*, 1986.
- [Lam58] J. Lambek. The mathematics of sentence structure. *Amer. Math. Monthly*, 65:154–169, 1958.
- [MCC00] R. Malouf, J. Carrol, and A. Copestake. Efficient feature structure operations without compilation. *Natural Language Engineering*, 1(1–18), 2000. Cambridge University Press.
- [MHF83] M. Marcus, D. Hindle, and M. Fleck. D-Theory: Talking about Talking about Trees. In *21st Annual Meeting of the Association for Computational Linguistics*, pages 129–136, 1983.
- [MK00] R. Muskens and E. Krahmer. Talking about trees and truth-conditions. In *Logical Aspects of Computational Linguistics, LACL’98. Grenoble, France, December 1998*, 2000.
- [Moo88] M. Moortgart. *Categorical Investigations: Logical and Linguistic Aspects of the Lambek Calculus*. Foris, Dordrecht, 1988.
- [Nas95] A. Nasr. A formalism and a parser for lexicalised dependency grammars. In *4th Int. Workshop on Parsing Technologies*. State University of NY Press, 1995.
- [Per00] G. Perrier. Interaction grammars. In *CoLing ’2000, Sarrebrücken*, pages 600–606, 2000.
- [Per02] G. Perrier. Descriptions d’arbres avec polarités : les grammaires d’interaction. In *9ième Conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN’02), Nancy, France, 2002*, pages 297–306, 2002.

- [Per04] G. Perrier. La sémantique dans les grammaires d'interaction. In *11ième Conférence annuelle sur le Traitement Automatique des Langues Naturelles (TALN 2004)*, Fès, Maroc, 2004.
- [PW83] F. Pereira and D. Warren. Parsing as deduction. In *Proc. of the 21st Annual Meeting of the Association for Computational Linguistics*, 1983.
- [RVS94] J. Rogers and K. Vijay-Shanker. Obtaining trees from their descriptions: an application to tree-adjoining grammars. *Computational Intelligence*, 10(4):401–421, 1994.
- [RVSW01] O. Rambow, K. Vijay-Shanker, and D. Weir. D-tree substitution grammars. *Computational Linguistics*, 27(1):87–121, 2001.
- [Shi85] S. Shieber. Using restriction to extend parsing algorithms for complex-feature-based formalisms. In *Proc. of the 23rd Annual Meeting of the Association for Computational Linguistics*, 1985.
- [SSP93] S. Shieber, Y. Schabes, and F. Pereira. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 12(1–199), 1993.
- [Tes34] L. Tesnière. Comment construire une syntaxe. *Bulletin de la Faculté des Lettres de Strasbourg*, 7 - 12^{ième} année:219–229, 1934.
- [VS92] K. Vijay-Shanker. Using description of trees in a tree adjoining grammar. *Computational Linguistics*, 18(4):481–517, 1992.
- [You67] D. Younger. Recognition and parsing of context-free languages in time n^3 . *Information and Control*, 10(2), 1967.