Natural Language Processing

Part 3: Syntax & grammar
chunking & constituents

Syntax

- Syntax defines the rules exploited to organize the words in a sentence on the basis of the following elements
  - Constituents – the syntax rules generally involve atomic tokens made up of a group of words (*chunks* that are considered as units at the syntax level). E.g, the **noun phrase** consists of groups made up of nouns, determiners, adjectives, conjunctions (the big house, a red and large carpet, …)
  - Grammatical relations – the represent the formalization of the sentence structure as a link between SUBJECTS and OBJECTS
    - es. [he]/SUBJECT took [the big hammer]/OBJECT
  - Subcategorizations and dependency relations – they are rules that express constraints between words and phrasal groups
    - e.g. want can be followed either by a verb infinite form or a noun phrase as object (I want to walk, I want a cake) whereas find can be followed only by a noun phrase (I found a treasure)

Constituents & chunking

- Generally one or more words can be grouped together to form a constituent or **chunk** that has a specific role in a sentence
  - A kind of constituent, e.g. a noun phrase NP, can only appear in given contexts, e.g. NP before a verb
    - [the flight from Paris] arrives late
    - [Philip's new car] is parked outside
  - Other kinds of constituents may have more than one admissible structure (preposed, postposed)
    - [On June 17th] I'll give the last lecture/I'll give the last lecture [on June 17th]
  - In any case the words composing a chunk are always organized as a unique group
  - Chunks can be modeled by **Context Free Grammars**
    - Chunking (**Shallow parsing**) algorithms group adjacent words into phrases

CF grammars & chunking

- The following rules can (partially) describe the structure of a **noun phrase** (NP)

  NP → Det Nominal

  NP → ProperNoun

  Nominal → Noun | Noun Nominal

  Det → a | the

  Noun → flight | plane | arrival

  - The lexicon (PoS tagger) is actually part of the rules since it assigns the corresponding PoS tag (the terminal symbol) to each word
- A verb phrase (VP) consists of a verb eventually followed by a sequence of other chunks such as a NP or a **prepositional phrase** (PP)

VP → Verb	*eat*
VP → Verb NP	*eat a pizza*
VP → Verb NP PP	*eat a pizza with the fork*
VP → Verb PP	*eat with the fork*

Chunks – PP & sentences

- A prepositional phrase (PP) is a words group that is generally started by a preposition

  PP → Preposition NP *with the white plastic fork*

  - The PP can have a very complex structure

- A simple model for a **complete sentence** (clause), representing the language start symbol S, is

  S → NP VP

  $[_S[_{NP}$My friend] $[_{Vp}$eats with the white plastic fork]]

Sentence level constructions

- There are many possible sentence structures for English. The main 4 structures are
  - declarative

    S → NP VP *The plane will land in the afternoon*
  - imperative

    S → VP *List all the flights to New York*
  - yes-no questions

    S → Aux NP VP *Will the plane arrive on time?*
  - wh-subject-question/wh-non-subject-question

    S → Wh-NP VP *Which flights depart at night?*

    S → Wh-NP Aux NP VP *Which flights does UA have from NYC?*

NP: Noun Phrase - 1

- A NP ha a **main noun** (head) with an optional set of modifiers that may appear before (prenominal or pre-head) or after (postnominal or post-head)
  - Generally a NP begins with a determiner (*a, the, this, any,..*) but it can also be omitted if the head noun is plural (e.g. *UA flights are based in..*) and is always omitted if the head noun is a mass noun (e.g. *water is a public resource*)
  - The NP can feature pre-determiners (*all the passengers*)
  - Between the determiner and the head noun tokens belonging to several word classes may appear (post-determiners)
    - cardinal numbers (*the two flights*) or ordinal numbers (*the first flight*) and quantifiers (*many flights*)
    - adjectives that can be grouped into an **adjective phrase** (AP) that may also include an adverb (*very expensive*)

NP: Noun Phrase - 2

- A simplified model for the NP can be defined considering the (optional) pre-head constituents

  NP → (Det) (Card) (Ord) (Quant) (AP) Nominal

- The main noun (head) can be followed by post-nominal modifiers
  - **Prepositional Phrase** (PP)

    Nominal → Nominal PP (PP) (PP) *flights from Rome to NYC by UA*

  - **Gerundive postnominals**– the head is followed by a verb phrase that begins with the gerundive form of the verb

    Nominal → Nominal GerundVP

    GerundVP → GerundV NP | GerundV PP *flights arriving from NYC*

    | GerundV | GerundV NP PP

NP: Noun Phrase - 3

- **non-finite clauses**– the noun is followed by a verbal phrase where the verb is in its infinite or past participle form
  - e.g. *the flights arrived last night, the aircraft used by this flight*
- **Relative clauses** – they often start with a relative pronoun (that, who,..) that is the subject of the embedded verb

Nominal → Nominal RelClause

RelClause → (who|that) VP

flights that arrive at night

  - A more complex case is when the relative pronoun has the role of object or complement in the relative close (e.g. *the steward, whom I asked the question to, the flight that I got on*)
- Post-nominal modifiers can be combined
  - *Is there a flight to NYC departing from Washington at night?*

Phrase/sentence coordination

- Noun phrases and other constituents can be coordinated with conjunctions (and, or, but,..)
  - e.g.

NP → NP and NP	*the pilots and the crew*

- Also verbal phrases and complete clauses can be coordinated

VP → VP and VP	*the flights departing from NYC and arriving in San Diego*
S → S and S	*I will fly to Denver and then I will drive to Aspen*

Gender/number agreement

- Agreement constraints can be modeled using grammar rules
  - An approach is to expand the grammar rules in order to detail all the agreement cases
    - e.g Questions in English for 3rd singular person/non 3rd singular person

S → Aux NP VP

S → 3sgAux 3sgNP VP
S → Non3sgAux Non3sgNP VP
3sgAux → does | has | can …
Non3sgAux → do | have | can …

  - The number of grammar rules increases since a given rule must be replicated for all the possible agreement combinations
    - This task can be faced with another approach, as for example by adding features to the grammar terminal symbols to exploit a more compact and readable formalism

Verb Phrase & Subcategories - 1

- A given verb may pose restrictions on the admissible syntactical structure/ kinds of complements that can appear in the verbal phrase
- A classic subcategorization is between **transitive** and **intransitive verbs**, but more recent grammar model propose up to about 100 subcategories
  - Subcategories are defined on the kinds of complements that can be associated to the given verb (**subcategorization frame**).
  - A verb can admit constructions corresponding to different frames

Frame	Verb	Example
⃠	sleep, walk	I walk
NP	find, take	I take a taxi
NP NP	show, give	I showed him my documents
PP_{from} PP_{to}	fly, go	I went from NYC to Boston
NP PP_{with}	help, load	He helped me with my baggage
VP_{to}	want, need	I need to leave
VP_{stem}	can, would	I can wait
S	mean, say	I say I will not go home

Verb Phrase & Subcategories - 2

- The constructions that are peculiar of each subcategory may be expressed by specific grammar rules (basically they are treated as syntactic constraints)

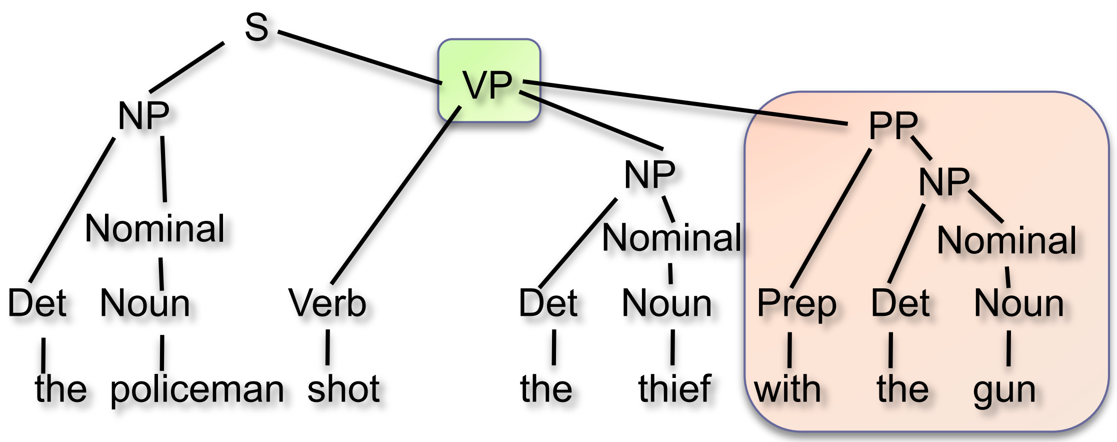
Verb-with-no-complement → walk | sleep |…	VP → Verb-with-no-complement
Verb-with-NP-complement → find | leave | ..	VP → Verb-with-NP-complement NP
Verb-with-S-Complement → think | say | …	VP → Verb-with-S-complement S
….	…..

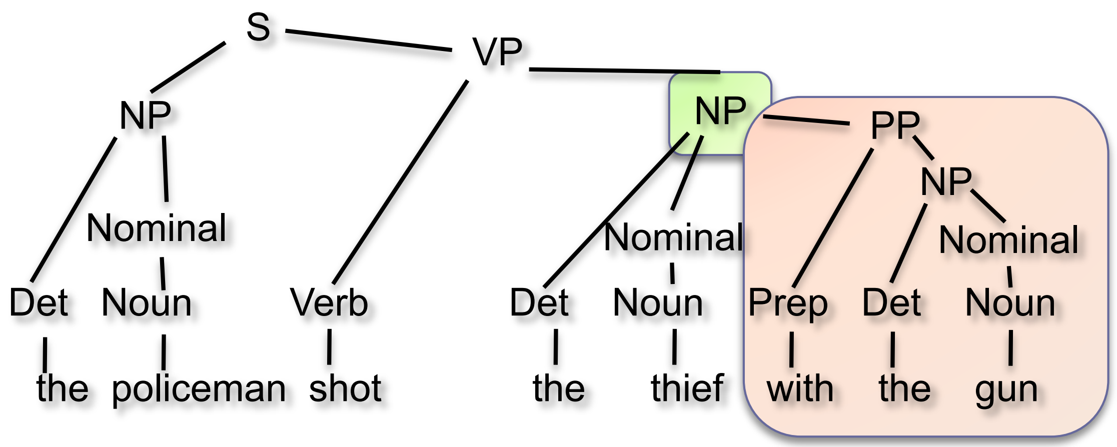
- This approach ha the disadvantage of requiring a specific rule for any potential construction increasing the number of grammar rules
- A more efficient solution is to exploit feature for the terminal symbols
- The need to organize words in subcategories based on constraints on the structure of the admissible constructions can be applied also to nouns, adjectives, and prepositions

CF parsing for NLP

- Grammars needed to model the syntactical structure of sentences in NLP may have features that make their parsing difficult
  - Left recursion
    - It creates problems for top-down parsers
    - It can be removed but the resulting equivalent grammar may model the syntactical structure in a unnatural way
  - Ambiguities
    - Structural ambiguity happens when there is more than one parse tree for a given sentence
    - An **attachment ambiguity** is present when a phrase can be placed in different positions in the parse tree
    - A **coordination ambiguity** is present when there are more constituents joined by coordinative conjunctions (and)

Ambiguities in NLP - attachment





The attachment ambiguity can not be solved at the syntactic level but only at the semantic/pragmatic level

Ambiguities in NLP - examples

Attachment ambiguity

I saw the Grand Canyon flying to New York

Can be solved at the semantic level…

.. the Grand Canyon does not fly!!

Coordination ambiguity

the bad boys and girls

two grouping are feasible

[the [bad boys] and [girls]]

[the [bad [boys and girls]]]

- Often the disambiguation to select only one among different parse trees can be realized only with statistical/semantic knowledge
  - Parsers not featuring a disambiguation module must yield all the possible parse trees
  - Yielding all the parse trees may be costly

The Earley algorithm (1970)

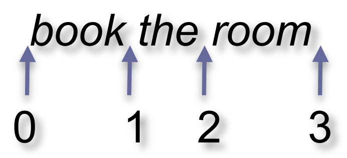
- The CF grammars exploited in NLP usually do not belong to the language subclasses for which an efficient parser can be obtained (e.g. LL(k) o LR(k))
  - Grammars for NLP are usually ambiguous and require to build all the possible parse trees for an input sentence
  - The **Earley algorithm** exploits dynamic programming to make the parse step efficient by storing all the partial parse subtrees corresponding to the sentence components in memory
    - It is a **parallel top-down parser** that avoids the repetition of the solution of the same sub-problems generated by the search with backtracking in order to reduce the complexity
    - In the worst case the algorithm has a $O(N^3)$ complexity where N is the number of words in the sentence
    - The algorithm executes a single scan from left to right filling an array (**chart**) of N+1 elements

The chart

- The **chart** stores efficiently the states visited while parsing
  - For each word in the sentence the chart contains the list of all the partial parse trees that have been generated up to a certain step
  - The chart stores a compact encoding of all the computed parse trees in the position corresponding to the sentence end
  - The parse subtrees are stored only once in the chart (the first time they are generated) and they are referred by pointers in the trees using them

  - Each state contains three elements
    - a subtree corresponding to a grammar rule
    - the degree of completion of the subtree (a dotted rule is used – LR(0) element)
    - The position of the subtree with respect to the input sequence encoded as a pair of indexes corresponding to the subtree start and to the dot position

States in the Chart



$S \rightarrow \bullet VP$, [0,0]

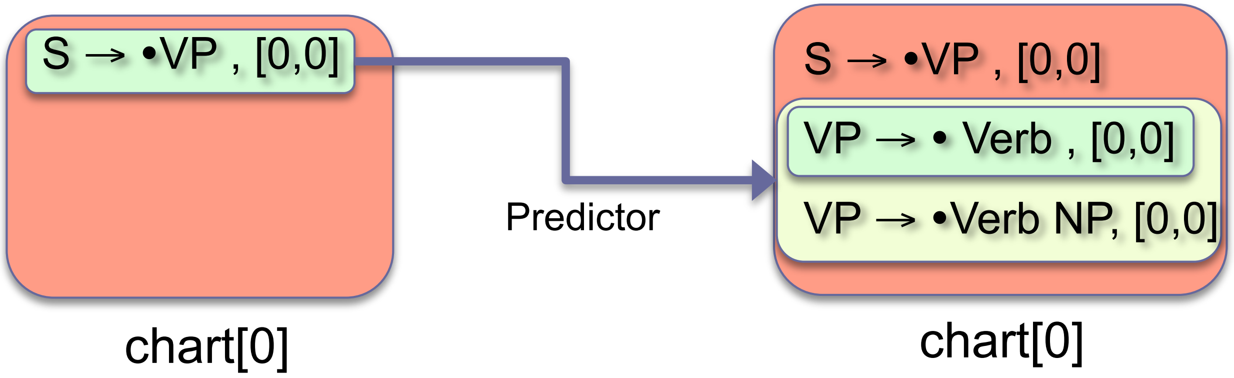
$NP \rightarrow Det \bullet Nominal$, [1,2]

$VP \rightarrow Verb\ NP \bullet$, [0,3]

- The parsing is performed by processing the states in the chart from left to right
  - At each step a state is selected and one out of three possible operations is applied. The operation may generate a new state in the current or following chart position
  - The algorithm always moves forward without removing the generated states but adding new states
  - The presence of the state $S \rightarrow \alpha \bullet$, [0,N] in the last chart position indicates a successful parse

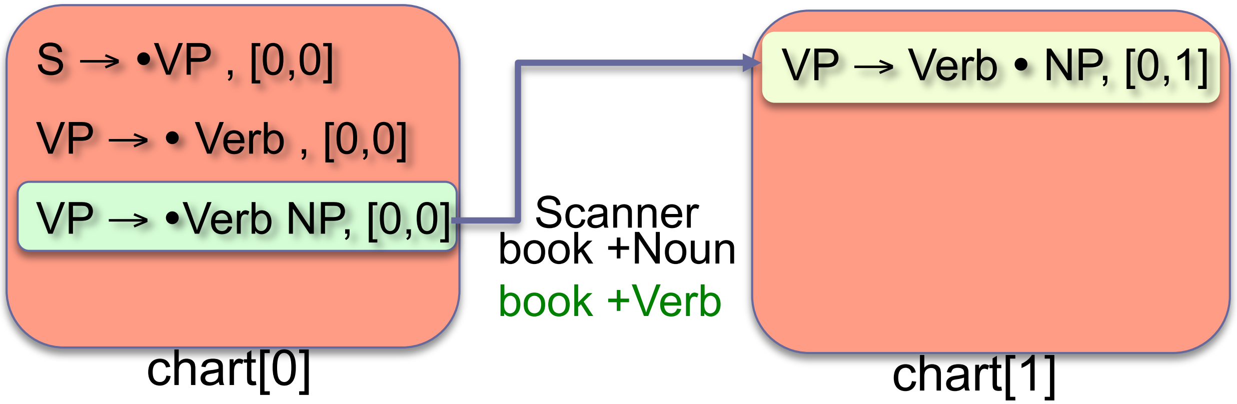
Parser operators - predictor

- **Predictor**
  - It creates new states base on the top-down parsing procedure
  - It is applied to each state that has a non terminal symbol on the right of the dot
  - A state is generated for any possible expansion of the non terminal symbol and it is inserted in the same chart position
  - The start and end positions for the inserted states are the same as those of the state that generated them



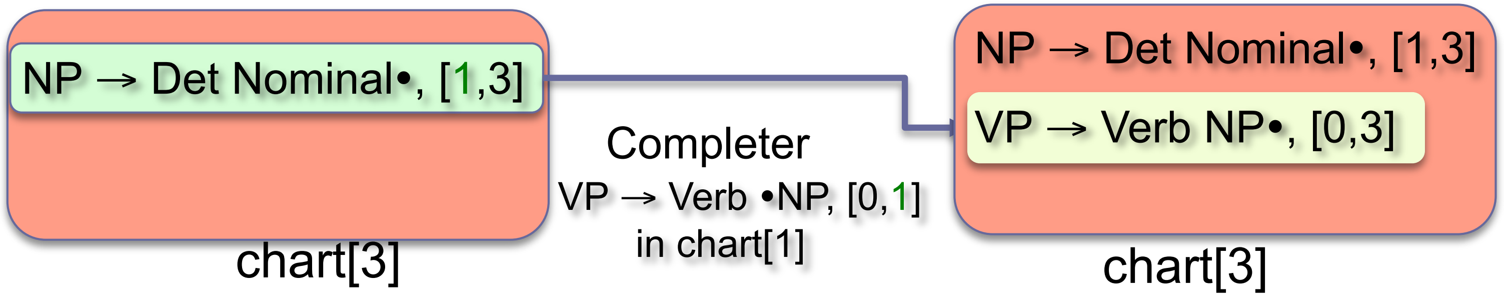
Parser operators - scanner

- **Scanner**
  - When a state has a PoS element on the right of the dot, the scanner checks the current symbol in the input sequence
  - A new state is created moving the dot on the right of the predicted PoS tag (the applied rule may also allow to disambiguate the word PoS if it is not unique)
  - The new state in inserted into the following chart position and the end position is increased by 1



Parser operators- completer

- **Completer**
  - It is applied to a state when the dot reaches the right end of the rule
  - It corresponds to the application of the production rule that explains a portion of the input with a syntactical category (non-terminal symbol)
  - The completer looks for all the states creates in the previous steps that where waiting for this non-terminal symbol
  - It adds all the found states to the current position moving the dot one position forward and adjusting the start and end positions based on the involved states



Building the parse tree

- The complete parse trees correspond to the states $S \rightarrow \alpha \bullet, [0,N]$ in the last chart position
  - The completions performed by the Completer module must be memorized to build the parse tree
  - For each state we need to keep track of the set of completed states that generated its components
  - This information can be added by the completer once a identifier is associated to each state (e.g. a serial number)
  - To build the parse tree the procedure starts from the complete rule at the chart position N tracing back recursively all the rewrite operations that have been stored

Features

- We can associate a set of features (properties) to each grammar element (terminal/non terminal symbol)
  - The features can be exploited to define a more compact representation of some kind of constraints (number/gender agreement, verb subcategories and related frames)
  - The features are property-value pairs where the value itself can be a structured entity

$$\begin{bmatrix} \textsc{Cat} & NP \\ \textsc{Number} & SG \\ \textsc{Person} & 3 \end{bmatrix}$$

feature for a flat representation of a 3sgNP

$$\begin{bmatrix} \textsc{Cat} & NP \\ \textsc{Agreement} & \begin{bmatrix} \textsc{Number} & SG \\ \textsc{Person} & 3 \end{bmatrix} \end{bmatrix}$$

feature for a structured 3sgNP representation

Feature unification

- **Unification** allows the fusion of two property structures in the case when they are compatible one with the other
  - It is a binary operator
  - A simple case is the following in which we check the number agreement

$$[\textsc{Number sg}] \sqcup [\textsc{Number sg}] = [\textsc{Number sg}]$$

$$[\textsc{Number sg}] \sqcup [\textsc{Number pl}] = \textsc{Fail}$$

$$[\textsc{Number sg}] \sqcup [\textsc{Number} \; []] = [\textsc{Number sg}]$$

  - if the feature has not an assigned valued for one of the operands, the feature is unified to the value assigned to the other operand and the fusion is successful

$$[\textsc{Number sg}] \sqcup [\textsc{Person} \; 3] = \begin{bmatrix} \textsc{Number} & \textsc{sg} \\ \textsc{Person} & 3 \end{bmatrix}$$

Subsumption

- The unification of two structure yield a feature structure that is more specific (or equal) with respect to the operands
  - It is more specific since it contains more assigned features
  - A less specific (more abstract) structure is said to **subsume** an equal or more specified structure

$$[\textsc{Number}\ \textsc{sg}] \sqsubseteq \begin{bmatrix} \textsc{Number} & \textsc{sg} \\ \textsc{Person} & 3 \end{bmatrix}$$

  - **Subsumption** defines a partial ordering (there are structures that are not subsumed one from the other)

$$[\textsc{Number}\ \textsc{sg}] \not\sqsubseteq [\textsc{Person}\ 3]$$
$$[\textsc{Person}\ 3] \not\sqsubseteq [\textsc{Number}\ \textsc{sg}]$$

Feature & grammar

- The grammar rules can be enriched with features
  - Feature structures can be assigned both to the lexical tokens (terminal symbols) and to syntactical elements (non terminal symbols)
  - The composition procedure defines how the feature structures associated to the elements in the production are combined to obtain the feature structure of the reduced symbol
  - Compatibility constraints are added

$$\begin{aligned} \beta_0 \quad & \rightarrow \beta_1 \dots \beta_n \\ & \{constraints\} \end{aligned}$$

"enriched" rules

$$\begin{aligned} \langle \beta_i feature \rangle &= value \\ \langle \beta_i feature \rangle &= \langle \beta_j feature \rangle \end{aligned}$$

kinds of constraints

- For example the number agreement can be expressed as

$$\begin{aligned} S \quad & \rightarrow NP\ VP \\ & < NP\ \textsc{Number} >=< VP\ \textsc{Number} > \end{aligned}$$

Feature & rules – example 1

- Agreement constraint in a noun phrase

$$\begin{aligned} NP &\rightarrow Det\ Nominal \\ &< Det\ \textsc{Agreement} >=< Nominal\ \textsc{Agreement} > \\ &< NP\ \textsc{Agreement} >=< Nominal\ \textsc{Agreement} > \end{aligned}$$

value unification

head features

$$\begin{aligned} Nominal &\rightarrow Noun \\ &< Nominal\ \textsc{Agreement} >=< Noun\ \textsc{Agreement} > \end{aligned}$$

$$\begin{aligned} Det &\rightarrow this & Noun &\rightarrow mouse \\ &< Det\ \textsc{Agreement Number} >= \textsc{sg} & &< Noun\ \textsc{Agreement Number} >= \textsc{sg} \\ Det &\rightarrow these & Noun &\rightarrow mice \\ &< Det\ \textsc{Agreement Number} >= \textsc{pl} & &< Noun\ \textsc{Agreement Number} >= \textsc{pl} \end{aligned}$$

Lexicon

Feature & rules – example 2

- Subcategorization (simplified version)
    - We need to define a category for each existing configuration (frame)

$$
\begin{aligned}
VP &\rightarrow Verb \\
&< VP\ \textsc{Head} > = < Verb\ \textsc{Head} > \\
&< VP\ \textsc{Head}\ \textsc{Subcat} > = \textsc{Intrans}
\end{aligned}
$$

head features

$$
\begin{aligned}
VP &\rightarrow Verb\ NP \\
&< VP\ \textsc{Head} > = < Verb\ \textsc{Head} > \\
&< VP\ \textsc{Head}\ \textsc{Subcat} > = \textsc{Trans}
\end{aligned}
$$

$$
\begin{aligned}
VP &\rightarrow Verb\ NP\ NP \\
&< VP\ \textsc{Head} > = < Verb\ \textsc{Head} > \\
&< VP\ \textsc{Head}\ \textsc{Subcat} > = \textsc{Ditrans}
\end{aligned}
$$

$$
\begin{aligned}
Verb &\rightarrow serves \\
&< Verb\ \textsc{Head}\ \textsc{Agreement}\ \textsc{Number} > = \textsc{sg} \\
&< Verb\ \textsc{Head}\ \textsc{Subcat} > = \textsc{Trans}
\end{aligned}
$$

Lexicon

Parsing & constraints

- The unification constraints on the features associated to the rule elements can be integrated directly in the parsing algorithm
  - The states that violate the constraints are removed
  - The Early algorithm can be modified to manage both the feature structures and the constraints associated to the grammar productions
    - The representation of the associated feature structure can be attached to each state
    - The Predictor module inserts a state that represents the constraints associated to the production rule
    - The Completer module combines two states (the completion of a production and the progression in another) and, hence, it must manage the unification of the two structures associated to the states considered to create the new one; if the two structure do not unify the Completer does not create the new state
    - A new state is not added if it is subsumed by a state already inserted in the chart (i.e. a more general state exists)

Probabilistic CF grammars

- Probabilistic parsing may provide hints to solve ambiguities
  - The parse tree having the highest probability is chosen
  - A stochastic CF grammar (**PCFG**) adds a conditional probability to each production rule

$$A \rightarrow \beta \ [p] \quad p = p(A \rightarrow \beta | A)$$

    - If we consider all the admissible expansions for the non terminal symbol A, the associated probabilities must sum to 1
  - A PCFG assigns a probability to each parse tree T
    - If we assume that the rules are independently chosen, the parse tree probability can be computed as

$$p(T) = p(T, S) = \prod_{n \in T} p(rule(n))$$

Probabilistic parsing

- The parse task for a PCF grammar is to yield the most likely parse tree fr a given input sentence

$$\hat{T}(S) = \underset{T \in \tau(S)}{\operatorname{argmax}}\, p(T)$$

  - The algorithms for the computation of the most likely parse tree are an extension of the standard parse algorithms (e.g. the Earley algorithm)
- Probabilities may be estimated form a corpus containing manually parsed sentences (treebank)
- PCFGs in their base form have limitations deriving form the independence assumption in the choice of the productions to be expanded
  - Actually there are structural and lexical dependencies
    - e.g. the way to expand a node may depend on its position in the parse tree (t.i. the subject of a clause is more likely to be a pronoun)

PCFG - limitations

- Another limitation of PCFGs is the independence from words
  - words have a crucial role in the selection of the correct parse tree in presence of ambiguities
  - Ambiguities in the coordinated elements are another example in which the lexical dependencies are important for disambiguation
    - e.g. dogs in houses and cats -> [dog in houses] and [cats]
- A solution is to associate the main word (head) of the functional part to the corresponding non-terminal symbol
  - Selection of the term on the right side of the production rule corresponding to the head (it is not always clear which one…)
  - Definition of a grammar with attributes (productions depend of the attribute value)
  - Independence assumptions to avoid the estimation of a probability for each head/rule in the lexicalized PCFG parser