<u>Left-corner parsing - a top-down method with bottom-up filtering</u>

(Analiza sintactica din coltul stang)

- **<u>Problems:</u>**
  - **<u>recursive descent parser</u> – goes into an infinite loop when it encounters a left-recursive production (because it applies the grammar productions blindly, without considering the actual input sentence).**
  - **<u>shift-reduce parser</u> – does not know how to treat null (missing) constituents**

- **<u>Solution:</u> a hybrid between these top-down and bottom-up approaches**

- **<u>The idea behind left-corner parsing</u>: accept a word from the input string, determine the type of constituent that starts with the corresponding category and parse the rest of that constituent in a top-down manner.**

- **<u>Remark</u>: the parse tree is “revealed” starting from the bottom left corner.**

Links

- **The left-corner parser can only partially deal with phrase-structure rules of type**

$$A \longrightarrow \phi$$

- **There are no problems when the parser encounters the null constituent while parsing top-down, namely in the situation**

$$A \text{ --> } B\ C$$

$$C \text{ --> } \phi$$

because constituent C will be analized (parsed) top-down. It is only necessary to specify to the parser (by means of phrase-structure rules) that, when looking for a C, it can go on without analyzing anything.

- **This does not hold for languages like Romanian, where a phrase-structure rule of type**

$$NP \rightarrow Det\ N$$

can be accompanied by the rule

$$Det \rightarrow \phi$$

"*o fata citeste*", "*fata citeste*", **namely when the null constituent is a null determinant occurring at the beginning of a noun phrase, a constituent that will be analyzed bottom-up.**

The group of rules

$$S \rightarrow NP\ VP$$

$$NP \rightarrow ART\ N$$

$$ART \rightarrow \phi$$

tells the parser to accept the null determinant as representing the first step in the parsing of a NP or of an S. In this case, the left-corner parser will get into a loop.

- **<u>The problem</u>: the described left-corner parser is allowed (just like the shift-reduce one) to accept null constituents anytime it wishes to do so.**

- **<u>Solution to the problem</u>: placing constraints on the parser by adding a <u>link table</u> that specifies <u>what *types* of constituents may occur on initial positions</u>. The link table is also called a <u>table of left corners</u>.**

- **By adding the table of left corners**
  - **the issue of accepting null constituents is solved (the parser can't accept them anytime);**
  - **the parser becomes more efficient.**

- **Implementation:**

Before starting its work, a left-corner parser preprocesses the context-free grammar to build a table where each row contains two cells, the first holding a non-terminal, and the second holding the collection of possible <u>left corners</u> of that non-terminal.

Category	**Left corners (pre-terminals)**
S	**NP**
VP	**V**

Corresponds to the phrase-structure rules

$$S \rightarrow NP\ VP$$

VP* → *V

VP* → *V NP

Storing the Intermediate Results: Chart Parsing

- **Storing intermediate results: a well-formed substring table (WFST) is a mechanism enabling a parser to keep a record of structures it has already found, so that it can avoid looking for them again.**

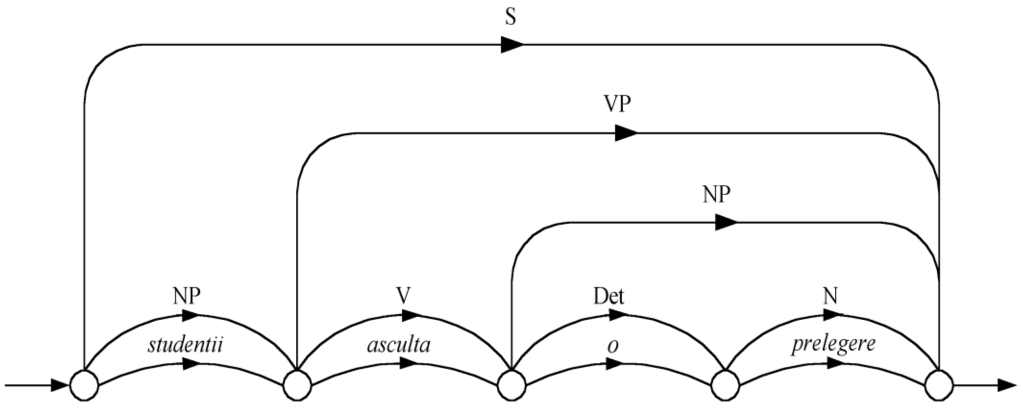
- **The extension represented by charts enables a parser, in addition, to record information about goals it has adopted. Recorded goals may have been unsuccessful or may still be under exploration. In either case it would be inefficient for the parser to start pursuing them again from scratch.**

- **With NLTK we apply the algorithm design technique of dynamic programming to the parsing problem. (Dynamming programming stores intermediate results and reuses them when appropriate, achieving significant efficiency gains). This approach to parsing is known as chart parsing.**

<u>Well-formed Substring Tables (WFST)</u>

- **Redundancy (repetition of the same situation) exists in the search space of a parsing algorithm. An efficient parser must have *<u>memory</u>*, namely it should be able to record the already found constituents and their structure.**

- **Consider the beginning and the end of the string to be numbered 0 and *n*, respectively, and the gaps between words to be numbered from 1 to *n-1* from left to right. A WFST tells us, for each pair of points *i, j* ($0 \leq i < j \leq$**



n), what categories can span the substring of words found between *i* and *j*.

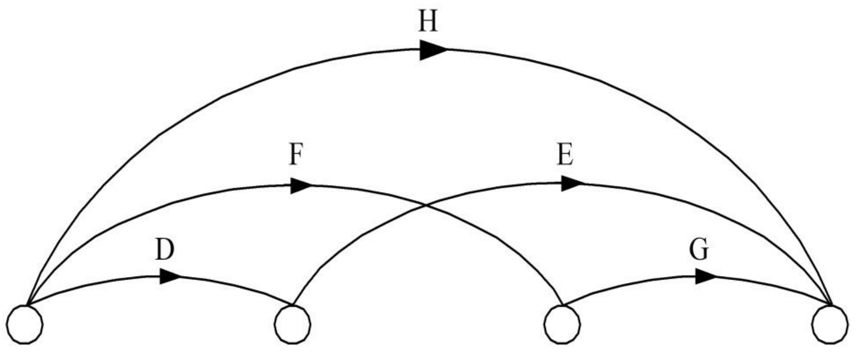
- **We think of a WFST as of a directed, acyclic graph with unique first and last nodes, a graph whose nodes are labelled from 0 (the first node) to *n* (the last), where *n* is the number of words in the string and whose arcs are labelled with syntactic categories and words.**

- **A WFST can be represented as a set of arcs, where an arc is a structure with the following attributes:**

  <START> = ... an integer ...
  <FINAL> = ... an integer ...
  <LABEL> = ... a category ...

Remarks :

- **A phrase structure tree is also a directed acyclic graph, but one in which the nodes are labelled with categories and the arcs are unlabelled.**
- **A WFST encodes the order of phrases directly. (This is in contrast to trees).**
- **Trees encode immediate dominance relations, indicating which phrases are parts of which other phrases while WFSTs do not. Simply by looking at the WFST, you can not be sure of the rule (or rules) that legitimates a**



particular arc. Example: Is the above covering H warranted in virtue of H dominating D and E or in virtue of H dominating F and G ?

- **This is a <u>neutral data structure</u>: they permit us to use any of bottom up, left to right etc. parsing techniques, in any combination and any degree.**

- **The parser refrains from rediscovering things that it has already found out. The parser always tests for the presence of a certain category in the WSTF before trying to form it.**

- **<u>Using a WFST:</u>**

A top-down parser, for instance, imposes the requirement that the already discovered syntactic groups are registered as entries in various locations of the WFST. When searching for a specific type of syntactic group, at a specific location in the input string, the table will be consulted. If the table contains an entry of the desired type, starting at the required position, then the table entries are used as possible analyses without the parser duplicating this work.

<u>The Active Chart</u> (Harta activa)

- **The use of a WFST will save work because it means that a successfully found syntactic group (for instance an NP) only needs to be found once. But it will not save the parser any time reinvestigating hypotheses that have previously failed.**

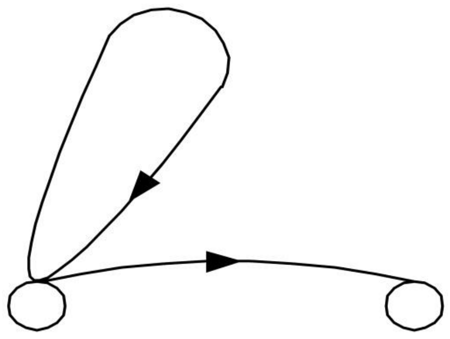
- **The WFST is a good way of representing facts about <u>structure</u> only. It does not help concerning structural hypotheses.**

- **The only way we can avoid duplicating previous <u>attempts</u> at parsing is to have an explicit representation about the different goals and hypotheses that the parser has at any one time.**

- **<u>Charts</u> will enable a parser, in addition, to record information about the goals it has adopted.**

We perform <u>two changes</u> to the data structure of a WFST:

- **Instead of requiring the directed graph to be strictly acyclic, we will relax things to permit <u>simple arcs</u> (called "*empty arcs*") that cycle back to the node they start out from. This will be the only type of cycle that will be permitted.**



- **The label on arcs changes from a simple category to a rule of the grammar. This rule will be called a "*dotted rule*", having the following significance:**

- **If *S → NP VP* is a rule of the grammar, then the following objects ("dotted rules") are well-formed arc labels:**

  S → .NP VP

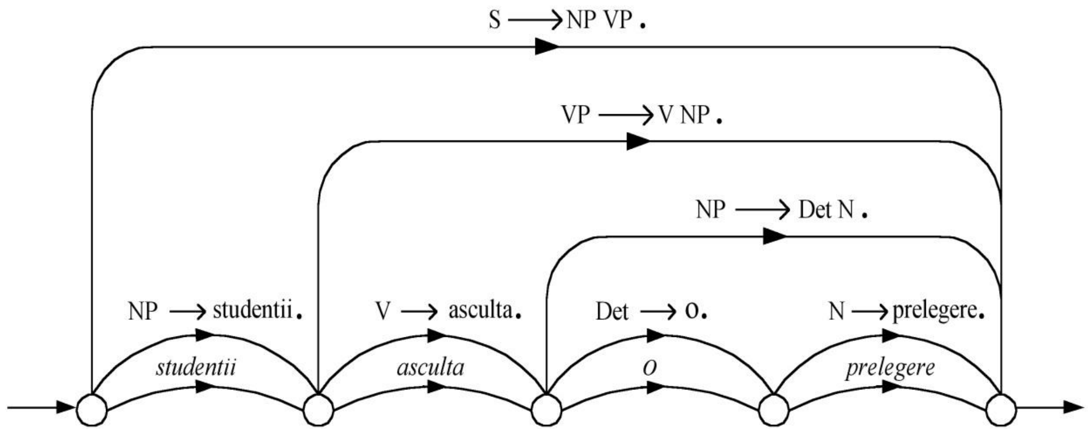
  S → NP.VP

  S → NP VP.

  <u>Significance</u>:

The 'dot' in one of these labels indicates to what extent the hypothesis that this rule is applicable has been verified by the parser.

<u>Example of a chart</u> for the same input string:



- **A WFST that has been modified in this way is known as an active chart (*harta activa)* and will be referred to simply as *chart*.**

- **The family of parsers that exploit the chart as a data structure are known as *chart parsers*.**

- **A node in a chart is referred to as a *vertex* and the arcs as *edges*.**

- **Arcs that represent unconfirmed hypotheses are known as *active edges* and those that represent confirmed hypotheses are known as *inactive edges*. Namely: an inactive edge represents a result, while an active edge represents a structure hypothesis (a hypothesis concerning structure).**

- **The chart is also completely neutral with respect to the chosen parsing strategy.**

CHART REPRESENTATION

- **We can represent a chart as a set of structures, each of which has the following attributes**

  < START > = < an integer >

  < FINISH > = < an integer >

  < LABEL > = < a category >

  < FOUND > = < a sequence of categories >

  < TOFIND > = < a sequence of categories >

where:

- **<LABEL> is the left hand side (LHS) of the appropriate dotted rule;**
- **<FOUND> is the sequence of right hand side (RHS) categories to the left of the dot;**
- **<TOFIND> is the sequence of RHS categories to the right of the dot.**

✓ **In this representation, an edge whose value for TOFIND is the empty sequence will be an inactive edge, and all other edges will be active.**

- **An alternative representation:**

<0, 2, S → NP.VP>

represents the following active and inactive edges:

<START> = 0

<FINISH> = 2

<LABEL> = S

<FOUND> = <NP>

<TOFIND> = <VP>

The Fundamental Rule of Chart Parsing

Fundamental rule: If the chart contains edges < *i, j, A* → *W1.B W2* > and < *j, k, B* → *W3.* >, *where A and B are categories and W1, W2 and W3* are (possibly empty) sequences of categories or words, then add edge < *i, k, A* → *W1 B.W2* > to the chart.

Note that the essence of chart parsing is the interaction between an active edge and an inactive edge of the desired category.

The result is either a new inactive edge or a new active edge representing an extension of the original active edge.

Effect: Put a new edge into the chart that spans both the active and inactive edges.

Note that the rule performs only additions of edges to the chart. This rule does not remove from the chart the active rules that were successful. This is useful for finding all possible parses of the input sentence.

Initialization (of the chart)

- **The task of ensuring that there are some inactive edges to be found is the job of initialization.**

In order to start parsing, namely to apply the fundamental rule, the chart must contain at least one active edge and one inactive edge. By initializing the chart we provide active edges as follows:

Assuming the following lexicon is given

Word studentii:

<cat> = NP.

Word asculta:

<cat> = V.

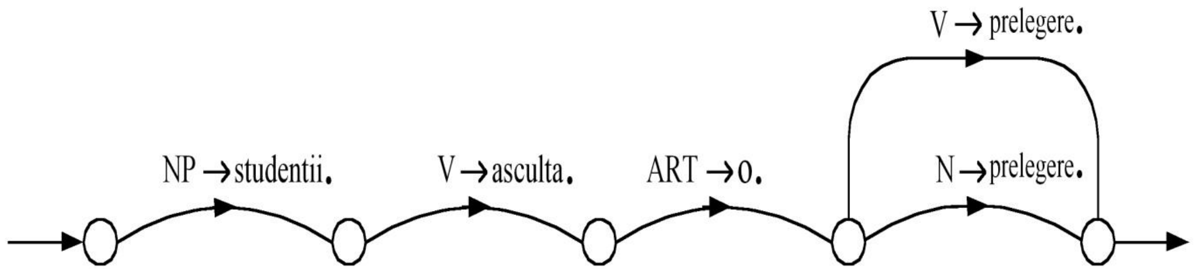
Word o:

<cat> = ART.

Word prelegere:

<cat> = N, V.

the corresponding initialized chart is the following:


V → prelegere.
NP → studentii.
V → asculta.
ART → o.
N → prelegere.

<u>Rule Invocation</u>

We still need a way of creating new <u>active edges</u> for the application of the fundamental rule!

There are two strategies for invoking rules: bottom-up and top-down

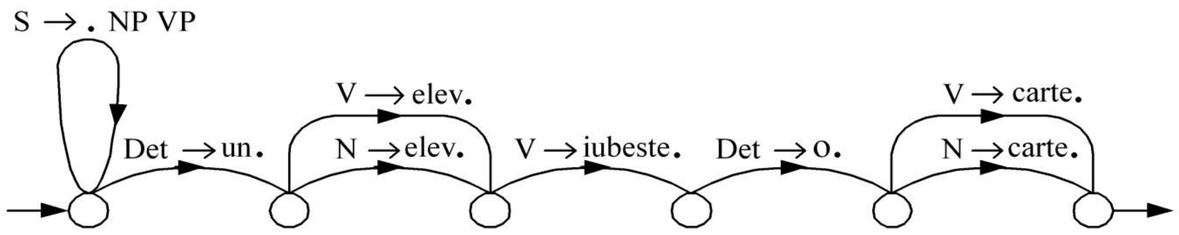
<u>Bottom-up rule</u>

If you are adding edge < *i, j, C* → *W1.* > to the chart, then for every rule in the grammar of the form *B* → *C W2*, add an edge < *i, i, B* →*.C W2* > to the chart.

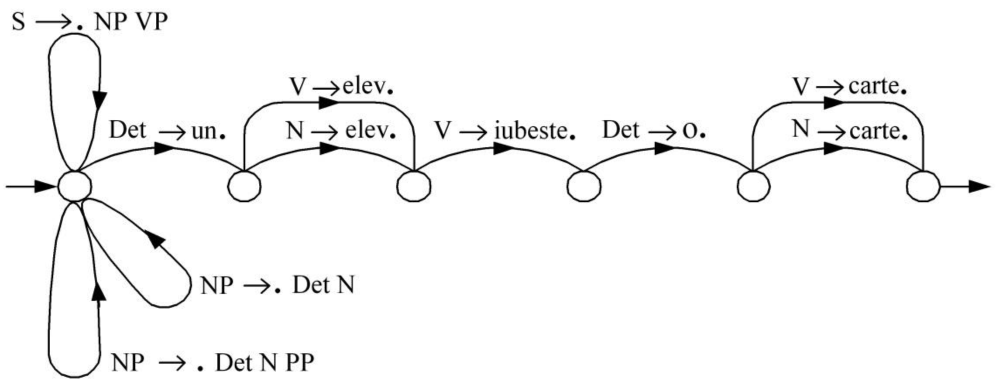
<u>Top-down strategy</u>

(1) **At initialization, for each rule *A* → *W*, where A is a category that can span a chart (typically S), add < *0, 0, A* →*.W*> to the chart.**
(2) **If you are adding edge < *i, j, C* → *W1.BW2*> to the chart, then for each rule *B* → *W*, add < *j, j, B* →*.W*> to the chart.**

<u>Example for the top-down strategy</u>; after the first clause:



After applying the second clause the chart becomes:



No matter what strategy we apply, we must take into account the fact that each active edge represents a hypothesis that must be explored. If this hypothesis is at least partially successful, then it can generate new hypotheses (both active and inactive edges), due to the action of the fundamental rule and the strategy for rule invocation that is applied.