Parsing with Context-Free Grammars

Berlin Chen

Department of Computer Science & Information Engineering
National Taiwan Normal University

References:

1. Natural Language Understanding, Chapter 3 (3.1~3.4, 3.6)
2. Speech and Language Processing (1st ed.), Chapters 9, 10; (3rd ed.), Chapter 17

Grammars and Sentence Structures (1/3)

- Describe the structure of sentences and explore ways of characterizing all the legal structures in a language
  - **Parse tree**: how a sentence is broken into its major subparts (constituents), and how these subparts are broken up in turn
  - E.g., *John ate the cat*

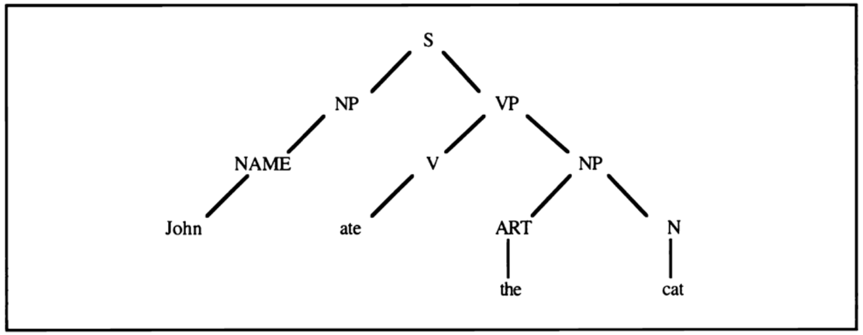


Figure 3.1 A tree representation of *John ate the cat*

```
(S (NP (NAME John))
   (VP (V ate)
   (NP (ART the)
      (N cat) )))
```

[list notation]

Rewrite Rules

1. S → NP VP	5. NAME → John
2. VP → V NP	6. V → ate
3. NP → NAME	7. ART → the
4. NP → ART N	8. N → cat

Grammar 3.2 A simple grammar

A set of rewrite rules describes what tree structures are allowable.

Formal models for capturing more sophisticated notions of grammatical structure, and algorithms for parsing these structures will be introduced.

Grammars and Sentence Structures (2/3)

- A grammar is said to derive a sentence if there is a sequence of rules that allow you to rewrite the start symbol S into the sentence
- A **top-down derivation strategy** starts with the **S** symbol and then searches through different ways to rewrite the symbols until the input sentence is generated

```
S

=> NP VP (rewriting S)
=> NAME VP (rewriting NP)
=> John VP (rewriting NAME)
=> John V NP (rewriting VP)
=> John ate NP (rewriting V)
=> John ate ART N (rewriting NP)
=> John ate the N (rewriting ART)
=> John ate the cat (rewriting N)
```

Grammars and Sentence Structures (3/3)

- In a **bottom-up derivation strategy**, you start with the words in the sentence and use the rewrite rules backward to reduce the sequence of symbols until it consists solely of S

```
=> NAME ate the cat (rewriting John)
=> NAME Vthe cat (rewriting ate)
=> NAME V ART cat (rewriting the)
=> NAME V ART N (rewriting cat)
=> NP V ART N (rewriting NAME)
=> NP V NP (rewriting ART N)
=> NP VP (rewriting V NP)
=> S (rewriting NP VP)
```

- Ideal properties of a grammar: **generality**, **selectivity**, **understandability**
  - **generality**, the range of sentences the grammar analyzes correctly; **selectivity**, the range of non-sentences it identifies as problematic; and **understandability**, the simplicity of the grammar itself

Recall: Context-Free Grammars (CFGs)

Noam Chomsky

- Grammars consist entirely of rules with a single symbol on the left-hand side
- Formalized by Chomsky (1956), and Backus (1959)
  - Also called Backus-Naur Form (BNF)
- Also called phrase-structure grammars
- A CFG defines a formal language
- The most commonly used mathematical system for modeling the constituent structure in natural languages
  - **Ordering**
    - What are the rules that govern the ordering of words and bigger units in the language
  - **Constituency**
    - How do words group into units and what do we say about how the various kinds of units behave

https://en.wikipedia.org/wiki/Noam_Chomsky

Major Characteristics of CFGs (1/4)

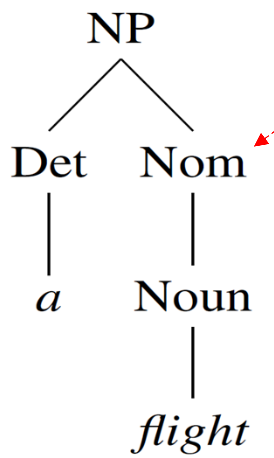
- CFG examples
  - Consist of a set of **rules** (**productions**)

Rewrite the symbol on the left with a string of symbols on the right.

NP → Det Nominal
NP → ProperNoum
Nominal → Noun | Noun Nominal
Det → a
Det → the
Noun → flight

mother

a nominal, in turn, can consist of one or more Nouns.



Nominal

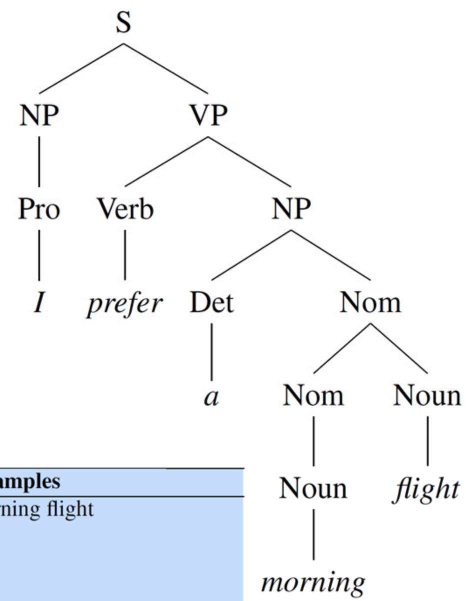
Generation:

So starting from the symbol: **NP**
we can use our first rule to rewrite **NP** as: **Det Nominal**
and then rewrite **Nominal** as: **Noun**
and finally rewrite these parts-of-speech as: **a flight**

When talking about these rules we can pronounce the right-arrow "→" as "goes to", and so we might read the first rule above as "NP goes to Det Nominal".

Major Characteristics of CFGs (2/4)

- Symbols used are divided into two classes: terminal and non-terminal symbols
  - A single **non-terminal symbols** on the left side of the arrow (→) while one or more terminal or non-terminal symbols on the right side
    - The **terminal symbol** is a word, while in **the lexicon**, the **non-terminal symbol** associated with each word is its lexical category, or part-of-speech (POS)
    - The **non-terminal symbol** can be a larger constituent (e.g. a phrasal unit) in addition to the lexical category



Grammar Rules			Examples
S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Nominal Noun*	morning + flight
	|	*Noun*	flights
VP	→	*Verb*	do
	|	*Verb NP*	want + a flight
	|	*Verb NP PP*	leave + Boston + in the morning
	|	*Verb PP*	leaving + on Thursday
PP	→	*Preposition NP*	from + Los Angeles

We can also represent a parse tree in a more compact format called **bracketed notation**:

$[_{S}\ [_{NP}\ [_{Pro}\ \text{I}]]\ [_{VP}\ [_{V}\ \text{prefer}]\ [_{NP}\ [_{Det}\ \text{a}]\ [_{Nom}\ [_{N}\ \text{morning}]\ [_{Nom}\ [_{N}\ \text{flight}]]]]]]$

Major Characteristics of CFGs (3/4)

- The notion of context in CFGs has nothing to do with the ordinary meaning of the **word context** in language
- All it really means that the non-terminal on the left-hand side of a rule is out of there all by itself

$$A \rightarrow B\ C$$

  - We can rewrite an *A* as a *B* followed by a *C* regardless of the context in which *A* is found

Major Characteristics of CFGs (4/4)

- **Generation** and **Parsing**: CFGs can be thought of as a device for *generating sentences* or a device for *assigning a structure to a given sentence (i.e. parsing)*

  - **Sentence generation**
    - Start from the *S* symbol, randomly choose and apply rewrite rules (or productions), until a sequence of words is generated

  - **Parsing**
    - Identify the structure of a sentence given a grammar
    - Top-down or bottom-up strategies

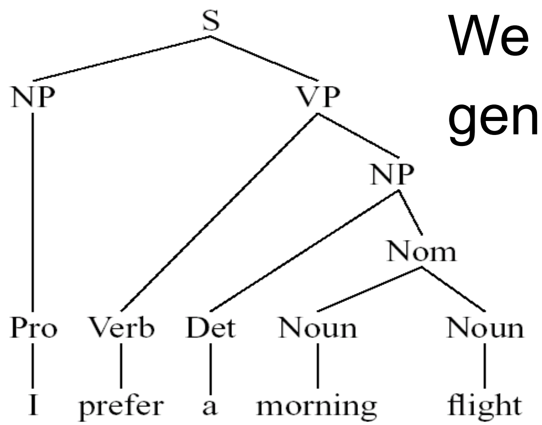
More Complex Derivations (1/3)

- *S* → *NP VP*
  - Units *S*, *NP*, and *VP* are in the language
  - *S* consists of an *NP* followed immediately by a *VP*
  - There may be many kinds of S
  - NPs and VPs can occur at other places (on the left sides) of the set of rules

    - E.g.

      NP → *Det Nominal*
      NP → *ProperNoum*
      VP → *Verb NP*
      VP → *Verb NP PP*

More Complex Derivations (2/3)



We can use the following grammar to generate sentences of this “ATIS-language”.

Bracketed notation (List notation)

$[_{S}\ [_{NP}[_{Pro}\ \mathbf{I}]][_{VP}[_{V}\ \mathbf{prefer}][_{NP}[_{Det}\ \mathbf{a}][_{Nom}\ [_{N}\ \mathbf{morning}][_{N}\ \mathbf{flight}]]]]]$

S	$\rightarrow$	$NP\ VP$	I + want a morning flight
NP	$\rightarrow$	$Pronoun$	I
	$\mid$	$Proper\text{-}Noun$	Los Angeles
	$\mid$	$Det\ Nominal$	a + flight
Nominal	$\rightarrow$	$Noun\ Nominal$	morning + flight
	$\mid$	$Noun$	flights
VP	$\rightarrow$	$Verb$	do
	$\mid$	$Verb\ NP$	want + a flight
	$\mid$	$Verb\ NP\ PP$	leave + Boston + in the morning
	$\mid$	$Verb\ PP$	leaving + on Thursday
PP	$\rightarrow$	$Preposition\ NP$	from + Los Angeles

$Noun$	$\rightarrow$	$flights \mid breeze \mid trip \mid morning \mid \ldots$
$\underline{Verb}$	$\rightarrow$	$is \mid \underline{prefer} \mid like \mid need \mid want \mid fly$
$Adjective$	$\rightarrow$	$cheapest \mid non-stop \mid first \mid latest$
		$\mid other \mid direct \mid \ldots$
$Pronoun$	$\rightarrow$	$me \mid I \mid you \mid it \mid \ldots$
$Proper\text{-}Noun$	$\rightarrow$	$Alaska \mid Baltimore \mid Los\ Angeles$
		$\mid Chicago \mid United \mid American \mid \ldots$
$Determiner$	$\rightarrow$	$the \mid a \mid an \mid this \mid these \mid that \mid \ldots$
$Preposition$	$\rightarrow$	$from \mid to \mid on \mid near \mid \ldots$
$Conjunction$	$\rightarrow$	$and \mid or \mid but \mid \ldots$

More Complex Derivations (3/3)

- Recursion
  - The non-terminal on the left also appears somewhere on the right (directly or indirectly)

    NP → NP PP   [[The flight] [to Boston]]

    VP → VP PP   [[departed Miami] [at noon]]

  - E.g.
    - flights from Denver (NP ⇒ NP PP)
    - Flights from Denver to Miami (NP PP ⇒ NP PP PP)
    - Flights from Denver to Miami in February (NP PP PP ⇒ NP PP PP PP)
    - Flights from Denver to Miami in February on Friday (NP PP PP PP ⇒ NP PP PP PP PP)

Formal Definition of Context-Free Grammar (1/3)

Capital letters like A, B, and S	Non-terminals
S	The start symbol
Lower-case Greek letters like α, β, and γ	Strings drawn from $(\Sigma \cup N)^*$
Lower-case Roman letters like u, v, and w	Strings of terminals

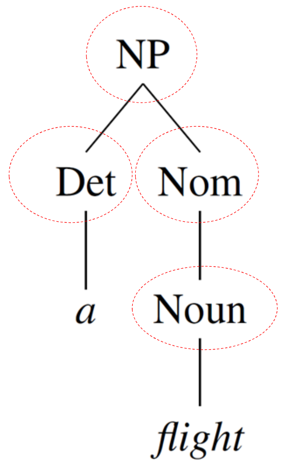
- A CFG *G* has four parameters ("4-tuple")
  1. A set of non-terminal symbols (or "variables") N
  2. A set of terminal symbols $\sum$ (disjoint from N)
  3. A set of rules (productions) *R*, each of the form $A \rightarrow \beta$, where A is a non-terminal symbol and β is a string of symbols from the infinite set of strings $(\sum \cup N)^*$
  4. A designated start symbol S (or N^1)

- CFG is a generative grammar
  - The language is defined by the set of possible sentences "**generated**" by the grammar
  - The concept of "**derivation**"
    - One string derives another one if it can be rewritten as the second one by **some series of rule applications**

Formal Definition of Context-Free Grammar (2/3)

- **Derivation**: a sequence of rules applied to a string that accounts for that string
  - The whole process can be represented by a parse tree
  - E.g. a parse tree for "a flight"

- But, usually languages are derivable from the designated start symbol (S)
  - The "sentence" node
  - The set of strings derivable from S called *sentences*



A derivation represented by a parse tree

Formal Definition of Context-Free Grammar (3/3)

- Directly Derive

  $A \rightarrow \beta$ is a production (rewrite) rule, where α and γ are any strings in$(\sum \cup N)^*$

  - Directly derive: $A \rightarrow \beta \quad \alpha A \gamma \Rightarrow \alpha \beta \gamma$

    where $\alpha, \beta, r, \alpha_i \in (\Sigma \cup N), m \geq 1$

  - Derive $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{m-1} \Rightarrow \alpha_m \quad \therefore \alpha_1 \overset{*}{\Rightarrow} \alpha_m$

    $\alpha_1, \alpha_2, \ldots, \alpha_m$ are any strings in$(\sum \cup N)^*$ $\quad$ (α_1 derives α_m)

- **Syntactic parsing:** the problem of mapping from a string of words to its parse tree(s) is called syntactic parsing

Treebanks

- A corpus in which every sentence is annotated with a parse tree is called a **treebank**
- Treebanks play an important role in parsing as well as in linguistic investigations of syntactic phenomena
- Treebanks are generally made by parsing each sentence with a parse that is then hand-corrected by human linguists
- The Penn Treebank project constructed various treebanks in English, Arabic, and Chinese

```
((S
   (NP-SBJ (DT That)
     (JJ cold) (, ,)
     (JJ empty) (NN sky) )
   (VP (VBD was)
     (ADJP-PRD (JJ full)
       (PP (IN of)
         (NP (NN fire)
           (CC and)
           (NN light) ))))
   (. .) ))
```

(a)

```
((S
   (NP-SBJ The/DT flight/NN )
   (VP should/MD
     (VP arrive/VB
       (PP-TMP at/IN
         (NP eleven/CD a.m/RB ))
       (NP-TMP tomorrow/NN )))))
```

(b)

Figure 17.5 Parses from the LDC Treebank3 for (a) Brown and (b) ATIS sentences.

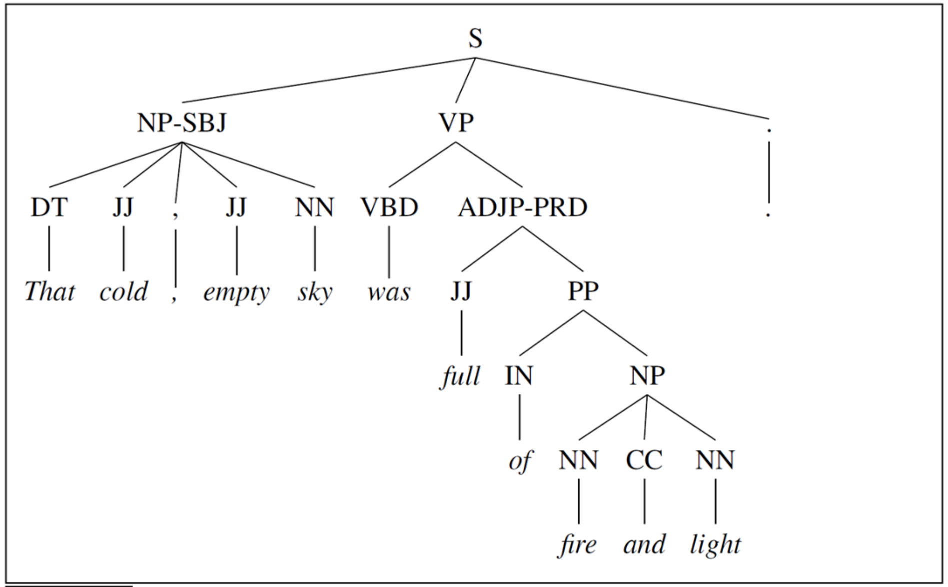


Figure 17.6 The tree corresponding to the Brown corpus sentence in the previous figure.

CFG: Conversion to Chomsky Normal Form (1/2)

- A context-free grammar is in Chomsky normal form (CNF) (Chomsky, 1963) if it is ϵ-free and if in addition each production is either of the form

$$A \rightarrow B\ C$$

or,

$$A \rightarrow a$$

- For CNF, the right-hand side of each rule either has two non-terminal symbols or one terminal symbol
- Chomsky binary normal form grammars are binary branching, that is they have binary trees (down branching to the **prelexical nodes**)

CFG: Conversion to Chomsky Normal Form (2/2)

- Any context-free grammar (CFG) can be converted into a weakly equivalent Chomsky normal form (CNF) grammar

$$A \rightarrow B\ C\ D \quad \Rightarrow \quad \begin{array}{l} A \rightarrow B\ X \\ X \rightarrow C\ D \end{array}$$

> Two grammars **are weakly equivalent** if they generate the same set of strings but do not assign the same phrase structure to each sentence.

Ambiguity (1/2)

- **Structural ambiguity** (viz. assigning more than one parse to a sentence) is the most serious problem faced by syntactic parsers
  - Two common kinds of ambiguity are **attachment ambiguity** and **coordination ambiguity**
  - A sentence has an attachment ambiguity if a particular constituent can be attached to the parse tree at more than one place.

attachment ambiguity:

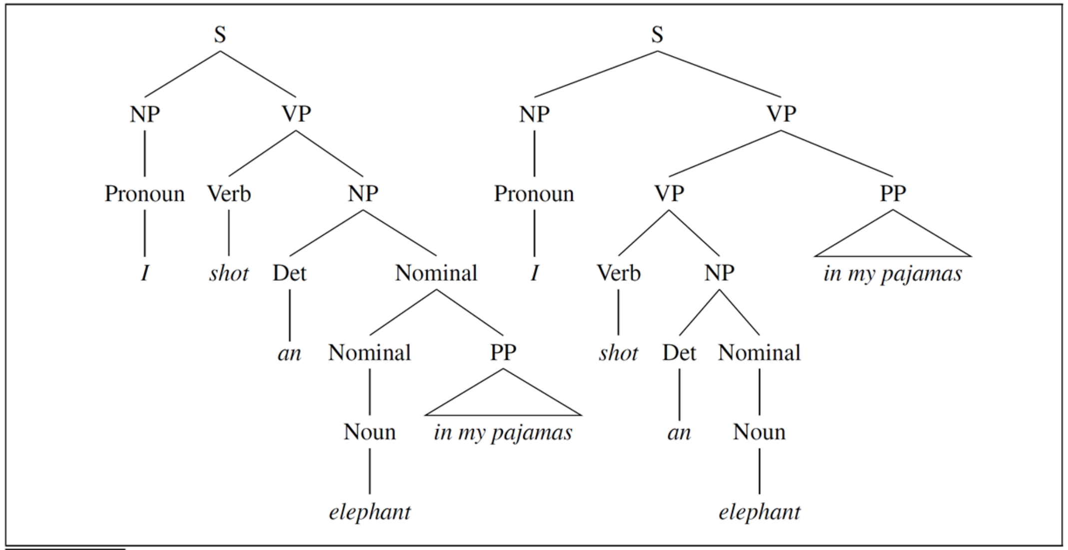


Figure 17.9 Two parse trees for an ambiguous sentence. The parse on the left corresponds to the humorous reading in which the elephant is in the pajamas, the parse on the right corresponds to the reading in which Captain Spaulding did the shooting in his pajamas.

coordination ambiguity:

[old [men and women]]

[old men] and [women]

Ambiguity (2/2)

- Various kinds of **adverbial phrases** are also subject to the attachment ambiguity, for example:

  We saw the Eiffel Tower flying to Paris.

  - The gerundive-VP flying to Paris can be part of a gerundive sentence whose subject is the Eiffel Tower or it can be an adjunct modifying the VP headed by saw

Lexical Head

- Syntactic constituents can be associated with a **lexical head**
  - For example, N is the head of an NP, V is the head of a VP
- In one simple model of lexical heads, each context-free rule is associated with a head
- The head is the word in the phrase that is grammatically the most important
- Heads are passed up the parse tree; thus, each non-terminal in a parse tree is annotated with a single word, which is its lexical head

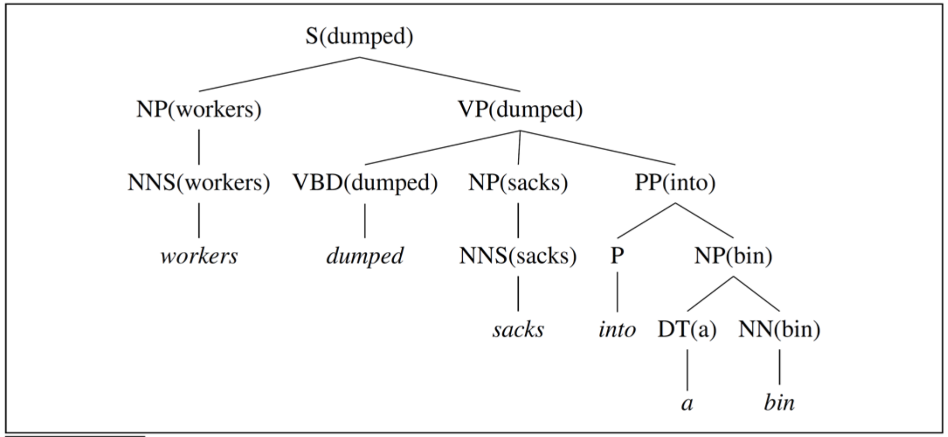


Figure 17.16 A lexicalized tree from Collins (1999).

Each CFG rule must be augmented to identify one right-side constituent to be the head child.

Sentence-level Construction of English

- **Declaratives**: A plane left.

  $S \rightarrow NP\ VP$

- **Imperatives**: Show the lowest fare.

  $S \rightarrow VP$

- **Yes-No Questions**: Did the plane leave?

  $S \rightarrow Aux\ NP\ VP$

- **WH Questions**: When did the plane leave?

  $S \rightarrow WH\ Aux\ NP\ VP$

Parsing Strategies (1/2)

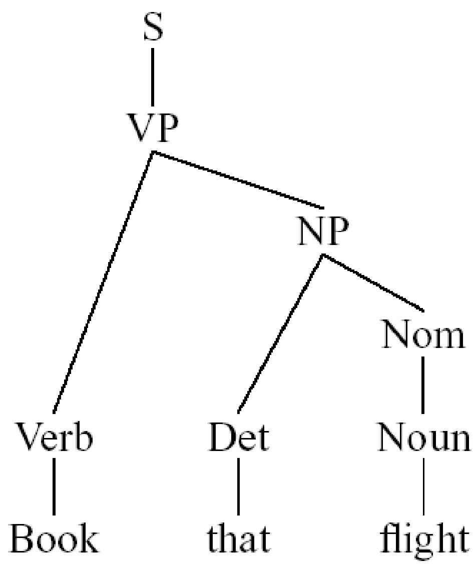
- Top-Down Parsing
  - Start with the *S* symbol and search through different ways to rewrite the symbols until the input sentence is generated, or until all possibilities have been explored

- Bottom-Up Parsing
  - Start with the words in the input sentence and use the rewrite rules backward to reduce the sequence of symbols until it consists solely of *S*
  - The left side of each rule is used to rewrite the symbols on the right side
    - Take a sequence of symbols and match it to the right side of the rule

Parsing Strategies (2/2)

Parsing as Search

- Different search algorithms, such as depth-first search (DFS) or breadth-first search (BFS) algorithms, can be applied



The record of the parsing process, either in top-down or bottom-up manners, can be used to generate the parse tree representation.

The Top-Down Parser

- Start with the *S* symbol and rewrite it into a sequence of terminal symbols that matches the classes of the words in the input sentence
  - The state of the parse at any given time can be represented as a list of symbols that are the results of operations applied so far

$_1$ **The** $_2$ **dog** $_3$ **cried** $_4$

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V
5. VP → V NP

Rewrite Rules

symbol lists

((N VP) 2)

((VP) 3)

((V) 3)

((V NP) 3)

The parser needs to find an N followed by a VP, starting at position 2

cried: V
dogs: N, V
the: ART

Lexicon

possible syntactic categories for each word

The Simple Top-Down Parser (1/3)

the possibilities list

Step	Current State	Backup States	Comment
1.	((S) 1)		initial position
2.	((NP VP) 1)		rewriting S by rule 1
3.	((ART N VP) 1)		rewriting NP by rules 2 & 3
		((ART ADJ N VP) 1)	
4.	((N VP) 2)		matching ART with *the*
		((ART ADJ N VP) 1)	
5.	((VP) 3)		matching N with *dogs*
		((ART ADJ N VP) 1)	
6.	((V) 3)		rewriting VP by rules 5–8
		((V NP) 3)	
	(( ) 4)	((ART ADJ N VP) 1)	
7.			the parse succeeds as V is matched to *cried*, leaving an empty grammatical symbol list with an empty sentence

new states are put onto the front of the possibilities list

Check the input sentence to see if a match (for lexical symbols) occurs ASAP

$_1$ The $_2$ dog $_3$ cried $_4$

Figure 3.5 Top-down depth-first parse of $_1$ *The* $_2$ *dogs* $_3$ *cried* $_4$

Rewrite Rules

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V
5. VP → V NP

Lexicon

cried: V
dogs: N, V
the: ART

The Simple Top-Down Parser (2/3)

- Algorithm

Step 1: Select the current state: take the first state off the possibilities list and call it **C**

- If the possibilities list is empty, then the algorithm fails

Step 2: If **C** consists of an empty symbol list and is at the sentence end position, the algorithm succeeds

Step 3: Otherwise, generate the next possible states

- If the first symbol on the symbol list is a lexical symbol (part-of-speech tag), and the next word in the sentence can be (matched) in that class, then create a new state by removing the first symbol from the symbol list and update the word position, and **add it to the possibilities list**
- Otherwise, if the first symbol on the symbol list of **C** is a non-terminal (but not the lexical symbol), generate a new state for each rule in the grammar that can rewrite that non-terminal symbol and **add them all to the possibilities list**

where to put the new states depends on the search strategies (e.g., BFS or DFS)

The Simple Top-Down Parser (3/3)

- One more example

$_1$ The $_2$ old $_3$ man $_4$ cried $_5$

Rewrite Rules

1. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V
5. VP → V NP

Lexicon

cried: V
old: ADJ, N
man: N, V
the: ART

new states are put onto the front of the possibilities list

the possibilities list

Step	Current State	Backup States	Comment
1.	((S) 1)		
2.	((NP VP) 1)		S rewritten to NP VP
3.	((ART N VP) 1)	((ART ADJ N VP) 1)	NP rewritten producing two new states
4.	((N VP) 2)	((ART ADJ N VP) 1)	
5.	((VP) 3)	((ART ADJ N VP) 1)	the backup state remains
6.	((V) 3)	((V NP) 3) ((ART ADJ N VP) 1)	
7.	(( ) 4)	((V NP) 3) ((ART ADJ N VP) 1)	
8.	((V NP) 3)	((ART ADJ N VP) 1)	the first backup is chosen
9.	((NP) 4)	((ART ADJ N VP) 1)	
10.	((ART N) 4)	((ART ADJ N) 4) ((ART ADJ N VP) 1)	looking for ART at 4 fails
11.	((ART ADJ N) 4)	((ART ADJ N VP) 1)	fails again
12.	((ART ADJ N VP) 1)		now exploring backup state saved in step 3
13.	((ADJ N VP) 2)		
14.	((N VP) 3)		
15.	((VP) 4)		
16.	((V) 4)	((V NP) 4)	
17.	(( ) 5)		success!

Figure 3.6 A top-down parse of $_1$ *The* $_2$ *old* $_3$ *man* $_4$ *cried* $_5$

(depth-first search)

Search strategies for the Top-Down Parser (1/3)

- Depth-first search: DFS (LIFO: last-in first-out)
  - The possibilities list is a stack
  - Step 1 always take the first element off the list
  - Step 3 always puts (adds) the new states **on the front of the list**

- Breadth-first search: BFS (FIFO: first-in first-out)
  - The possibilities list is a queue
  - Step 1 always take the first element off the list
  - Step 3 always puts (adds) the new states **on the end of the list**

Search strategies for the Top-Down Parser (2/3)

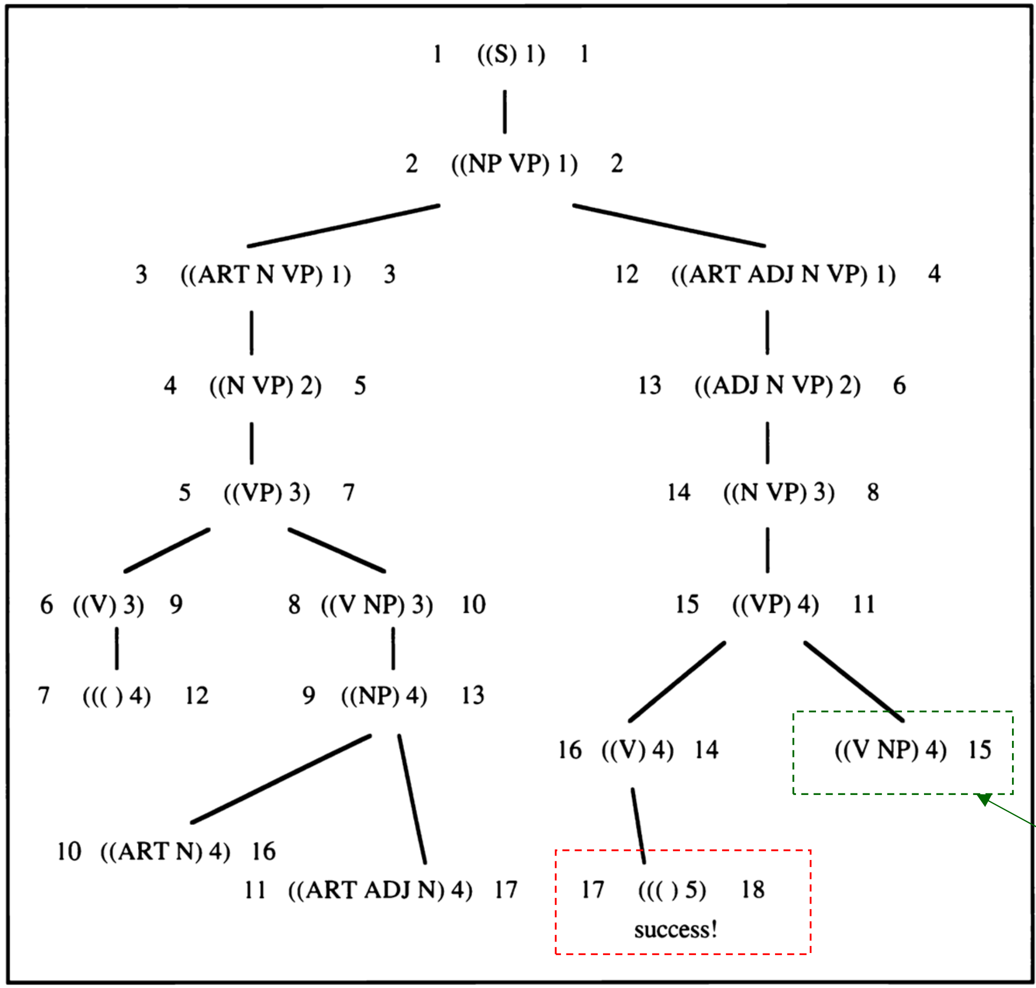


Figure 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

$_1$ The $_2$ old $_3$ man $_4$ cried $_5$

The number beside each node records when the node was selected to be processed by the algorithm.
On the left side is the order produced by the **depth-first strategy (DFS),** and on the right side is the order produced by **the breadth-first strategy (BFS)**.

Search strategies for the Top-Down Parser (3/3)

- Comparison of DFS and BFS
  - DFS
    - **One interpretation** is considered and expanded until fails; only then is the second one considered
    - Often moves quickly to the a solution but in other cases may spend considerable time pursuing futile paths
  - BFS
    - **All interpretations** are considered alternatively, each being expanded one step at a time
    - Explore each possible solution to a certain depth before moving on

Many parsers built today use the DFS strategy because it tends to minimize the no. of backup states needed and thus uses less memory and requires less bookkeeping.

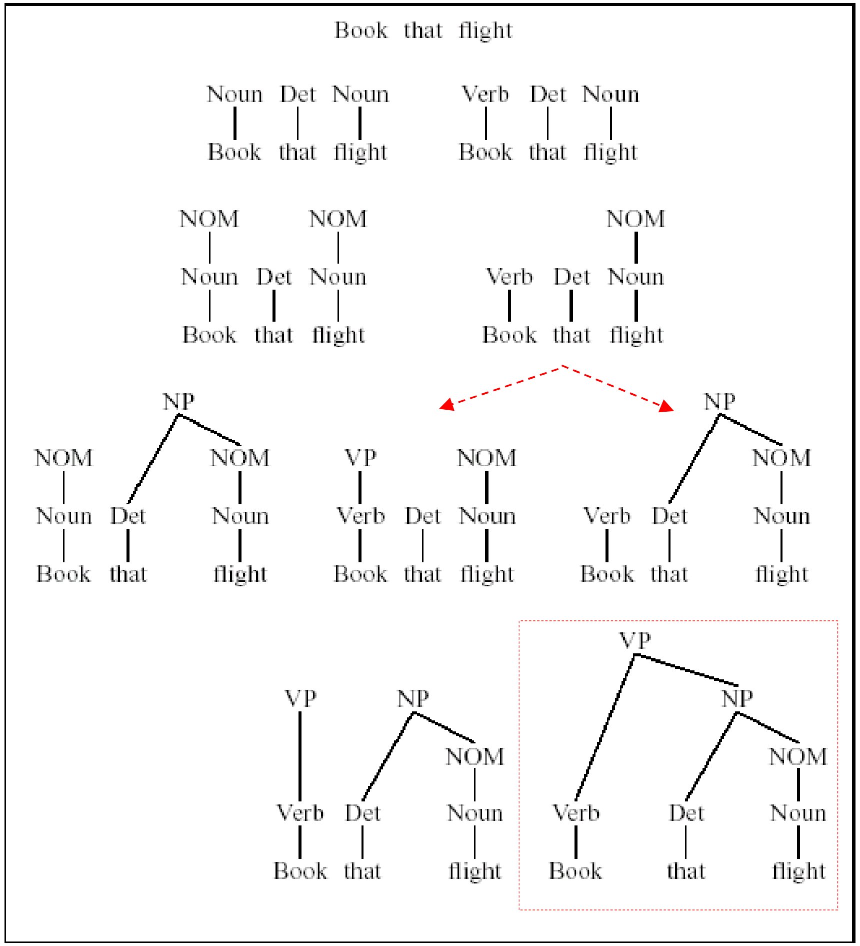
The Bottom-Up Parser (1/2)

- Start with the words of the input, and try to build tree from the words up, by applying rules from the grammar one at a time
  - The right hand side of some rules might fit
  - Successful if the parser succeeds in building a tree rooted in the start symbol (or a symbol list with S and positioned at the end of the input sentence) that covers all the input

The Bottom-Up Parser (2/2)

$S \rightarrow NP\ VP$	$Det \rightarrow that \mid this \mid a$
$S \rightarrow Aux\ NP\ VP$	$Noun \rightarrow book \mid flight \mid meal \mid money$
$S \rightarrow VP$	$Verb \rightarrow book \mid include \mid prefer$
$NP \rightarrow Det\ Nominal$	$Aux \rightarrow does$
$Nominal \rightarrow Noun$	
$Nominal \rightarrow Noun\ Nominal$	$Prep \rightarrow from \mid to \mid on$
$NP \rightarrow Proper\text{-}Noun$	$Proper\text{-}Noun \rightarrow Houston \mid TWA$
$VP \rightarrow Verb$	
$VP \rightarrow Verb\ NP$	$Nominal \rightarrow Nominal\ PP$

$_1$ Book $_2$ that $_3$ flight $_4$



Comparing Top-Down and Bottom-UP Parsing

- Top-Down
  - Pro: Never wastes time exploring trees that cannot result in an ***S***
  - Con: But spends considerable effort on ***S*** trees that are not consistent with the input
- Bottom-Up
  - Pro: Never suggest trees that are not least locally grounded in the actual input
  - Con: Trees that have no hope of leading to an S, or fitting in with any of their neighbors, are generated with wild abandon
  - Pro: Only check the input once

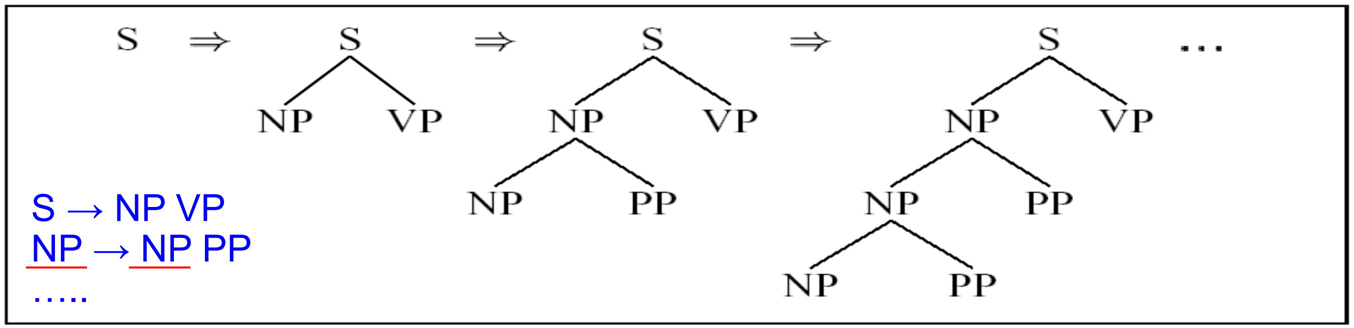
Problems with Parsing (1/4)

- **Left-recursion**
  - A non-terminal category that has a derivation that includes itself anywhere along its leftmost branch

    NP → Det   Nominal

    Det → NP ’s
    - Especially, **the immediately** left-recursive rule

      NP → NP   ’s   N
      - E.g. causing a infinite loop in top-down parsing with DFS search strategy



Problems with Parsing (2/4)

- **Ambiguity**
  - Structural ambiguity: arises in the syntactic structures used in parsing
    - The grammar assigns more than one possible parse to a sentence
      - Attachment ambiguity: Most frequently seen for adverbial phrases (PP-attachment ambiguity)

        I shot an elephant in my pajamas.

      - Coordination ambiguity

        old men and women

Parsers which do not incorporate disambiguators must simply return all the possible parse trees for a given input.

Problems with Parsing (3/4)

- **Ambiguity**

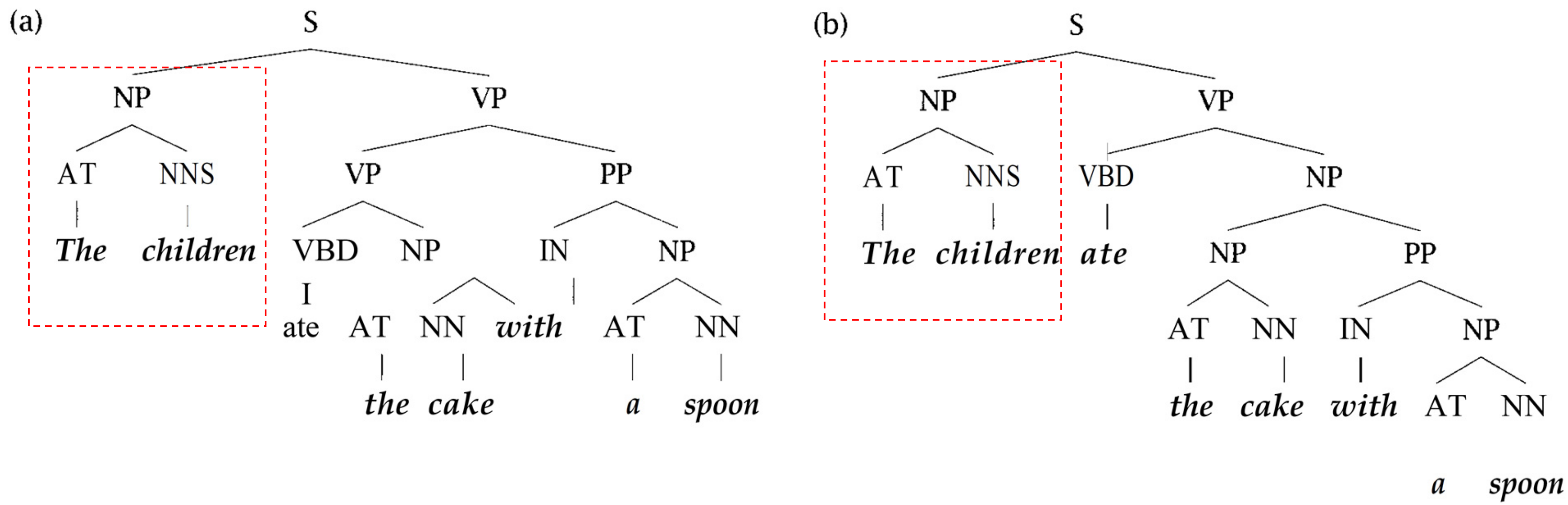
  Basic ways to alleviate the ambiguity problem

  - Dynamic programming
    - Used to exploit the regularities in the search space so that the common subpart are derived only once
      - Reduce some of the costs associated with ambiguity
    - Implicitly store all possible parses by storing all the constituents with links that enable the parses to be reconstructed

  - Heuristic search
    - Augment the parser's search strategy with heuristics that guide it towards likely parses first

Problems with Parsing (4/4)

- Repeated Parsing of Subtrees
  - The parser often builds valid subtrees for portions of the input, then discards them during the backtracking
    - It has to rebuild these subtrees again
    - Some constituents are constructed more than once



An example of a prepositional phrase attachment ambiguity.

CKY Parsing (1/2)

- The **dynamic programming** advantage arises from the context-free nature of our grammar rules
  - Once a constituent has been discovered in a segment of the input we can record its presence and make it available for use in any subsequent derivation that might require it
- The Cocke-Kasami-Younger (CKY) algorithm is the most widely used dynamic-programming based approach to parsing
  - **Chart parsing** (Kaplan 1973, Kay 1982) is a related approach
  - Dynamic programming methods are often referred to as chart parsing methods

CKY Parsing (2/2)

- The CKY algorithm requires grammars to first be in Chomsky Normal Form (CNF), namely being binary branching
  - Recall that CNF: A → B C or A → w
- The entire conversion process can be summarized as follows:
  1. Copy all conforming rules to the new grammar unchanged
  2. Convert terminals within rules to dummy non-terminals

     INF-VP → to VP ⟹ INF-VP → TO VP, TO → to

  3. Convert unit productions

     Nominal → Noun ⟹ Nominal → book | flight | meal | …

  4. Make all rules binary and add them to new grammar

$\mathcal{L}_1$ **Grammar**	$\mathcal{L}_1$ **in CNF**
$S \rightarrow NP\ VP$	$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$	$S \rightarrow X1\ VP$
	$X1 \rightarrow Aux\ NP$
$S \rightarrow VP$	$S \rightarrow book \mid include \mid prefer$
	$S \rightarrow Verb\ NP$
	$S \rightarrow X2\ PP$
	$S \rightarrow Verb\ PP$
	$S \rightarrow VP\ PP$
$NP \rightarrow Pronoun$	$NP \rightarrow I \mid she \mid me$
$NP \rightarrow Proper\text{-}Noun$	$NP \rightarrow TWA \mid Houston$
$NP \rightarrow Det\ Nominal$	$NP \rightarrow Det\ Nominal$
$Nominal \rightarrow Noun$	$Nominal \rightarrow book \mid flight \mid meal \mid money$
$Nominal \rightarrow Nominal\ Noun$	$Nominal \rightarrow Nominal\ Noun$
$Nominal \rightarrow Nominal\ PP$	$Nominal \rightarrow Nominal\ PP$
$VP \rightarrow Verb$	$VP \rightarrow book \mid include \mid prefer$
$VP \rightarrow Verb\ NP$	$VP \rightarrow Verb\ NP$
$VP \rightarrow Verb\ NP\ PP$	$VP \rightarrow X2\ PP$
	$X2 \rightarrow Verb\ NP$
$VP \rightarrow Verb\ PP$	$VP \rightarrow Verb\ PP$
$VP \rightarrow VP\ PP$	$VP \rightarrow VP\ PP$
$PP \rightarrow Preposition\ NP$	$PP \rightarrow Preposition\ NP$

Figure 17.10 $\mathcal{L}_1$ Grammar and its conversion to CNF. Note that although they aren't shown here, all the original lexical entries from $\mathcal{L}_1$ carry over unchanged as well.

CYK Recognition (1/4)

- Employ the CYK algorithm to tell whether a valid exists for a given sentence based on whether or not CYK finds an S in cell [0, *n*] of the **parse table** we maintain for a sentence of length *n*

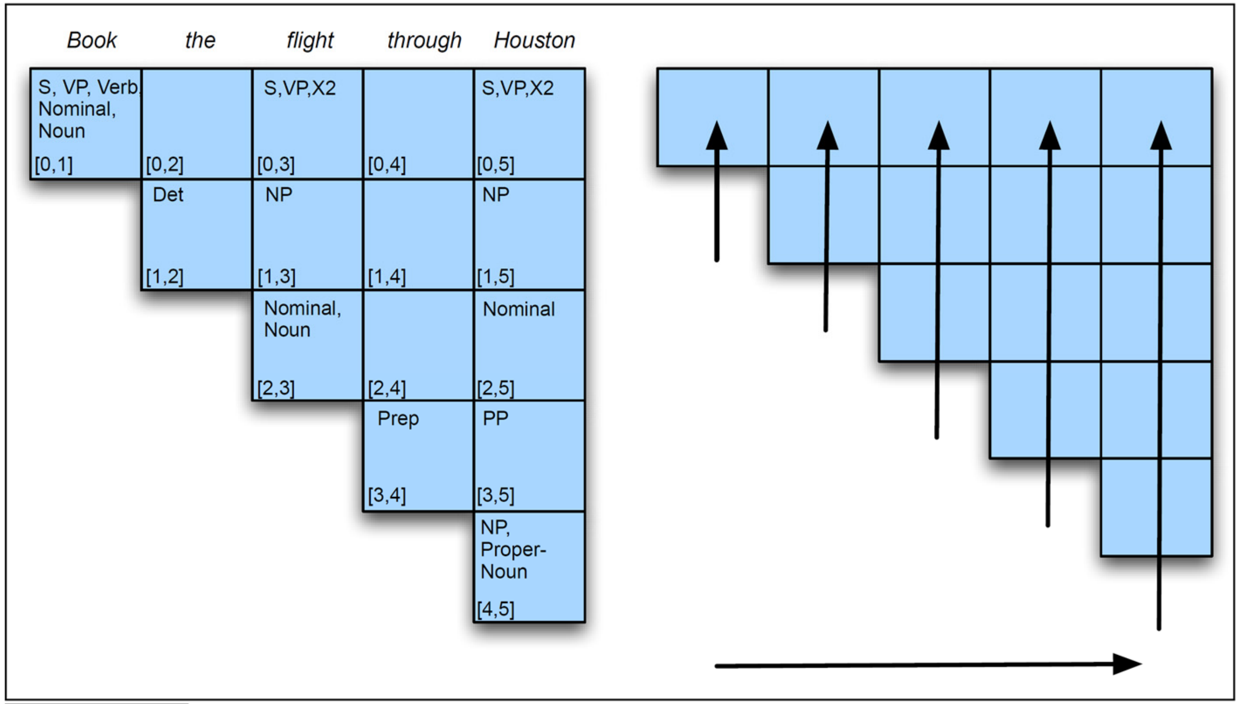


Figure 17.11 Completed parse table for *Book the flight through Houston.*

$\text{Book}_1\ \text{the}_2\ \text{flight}_3\ \text{through}_4\ \text{Houston}_5$

CYK Recognition (2/4)

- The CKY algorithm

Complexity: $O(N^3L)$?

function CKY-PARSE(*words, grammar*) **returns** *table*

for $j \leftarrow$ **from** 1 **to** LENGTH(*words*) **do**
  for all $\{A \mid A \rightarrow words[j] \in grammar\}$
    $table[j-1, j] \leftarrow table[j-1, j] \cup A$ Filling POS tags (smallest constituents)
  for $i \leftarrow$ **from** $j-2$ **down to** 0 **do**
    for $k \leftarrow i+1$ **to** $j-1$ **do**
      for all $\{A \mid A \rightarrow BC \in grammar$ **and** $B \in table[i,k]$ **and** $C \in table[k,j]\}$
        $table[i,j] \leftarrow table[i,j] \cup A$ Filling larger constituents

Figure 17.12 The CKY algorithm.

- The outermost loop of the algorithm iterates over the columns, from left to right
- The second loop iterates over the rows, from the bottom up
  - At each such split *k*, the algorithm considers whether the contents of the two cells can be combined in a way that is sanctioned by (被認可) a rule in the grammar

CYK Recognition (3/4)

Book$_1$ the$_2$ flight$_3$ through$_4$ Houston$_5$

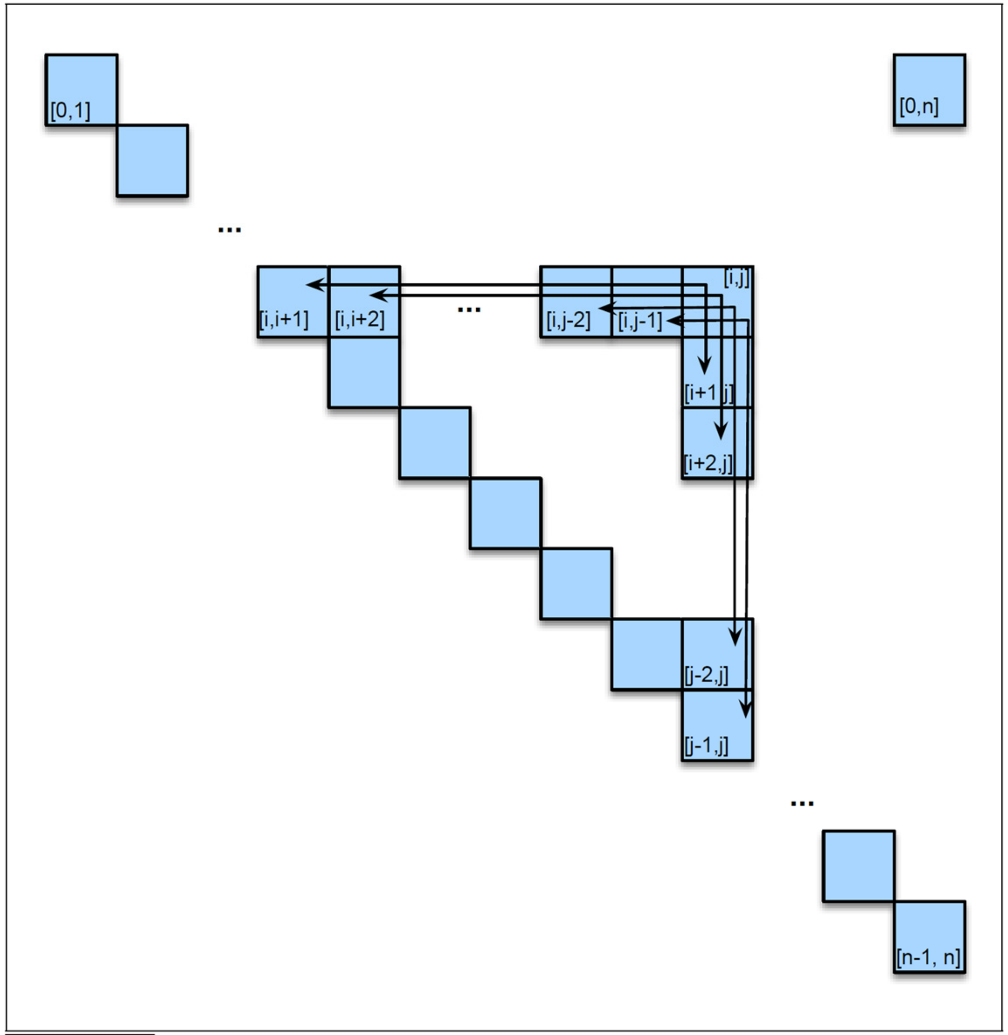


Figure 17.13 All the ways to fill the [*i*, *j*]th cell in the CKY table.

CYK Recognition (4/4)

- This figure shows how the five cells of column 5 of the table are filled after the word Houston is read

Book$_1$ the$_2$ flight$_3$ through$_4$ Houston$_5$

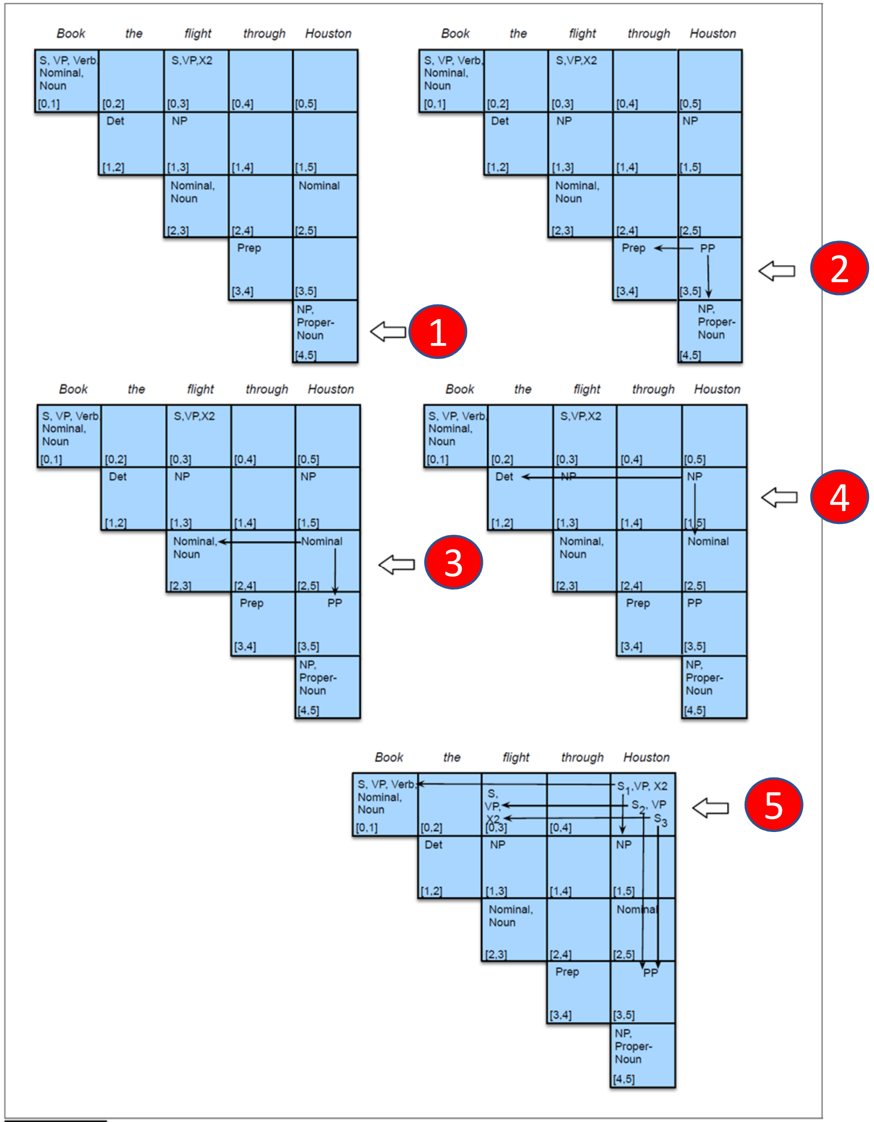


Figure 17.14 Filling the cells of column 5 after reading the word *Houston*.

Change CKY Recognition to CKY Parsing

- To turn **CKY recognition** into a **parser** capable of returning all possible parses for a given input, we can make two simple changes to the algorithm

    1. The first change is to augment the entries in the table so that each non-terminal is paired with pointers to the table entries from which it was derived
    2. The second change is to permit multiple versions of the same non-terminal to be entered into the table

Span-Based Neural Constituency Parsing (1/4)

- **Neural CKY** (Kitaev et al., 2018, 2019)
  Berkeley
  - Train a neural classifier to assign a score to each constituent
    - Introduce a parser that combines an **encoder** built using this kind of self-attentive architecture with a **decoder** customized for **chart parsing**
  - Then, use a modified version of CKY to combine these constituent scores to find the best-scoring parse tree
- More properties of **Neural CKY**
  - Learns to map a span of words to a constituent, and, like CKY, hierarchically combines larger and larger spans to build the parse-tree bottom-up
  - But unlike CKY, this parser does not use the hand-written grammar to constrain what constituents can be combined, instead just relying on the learned neural representations of spans to encode likely combinations

1. N. Kitaev and D. Klein, "Constituency parsing with a self-attentive encoder," ACL 2018
2. N. Kitaev et al., "Multilingual constituency parsing with self-attention and pre-training ," ACL 2019

Span-Based Neural Constituency Parsing (2/4)

For a parse tree T

$T = \{(i_t, j_t, l_t) : t = 1, \dots, |T|\}$

- Schematic Depiction $s(i, j, l)$

non-terminal label (e.g., NP)

$s(T) = \sum_{(i,j,l) \in T} s(i, j, l)$

non-terminal label

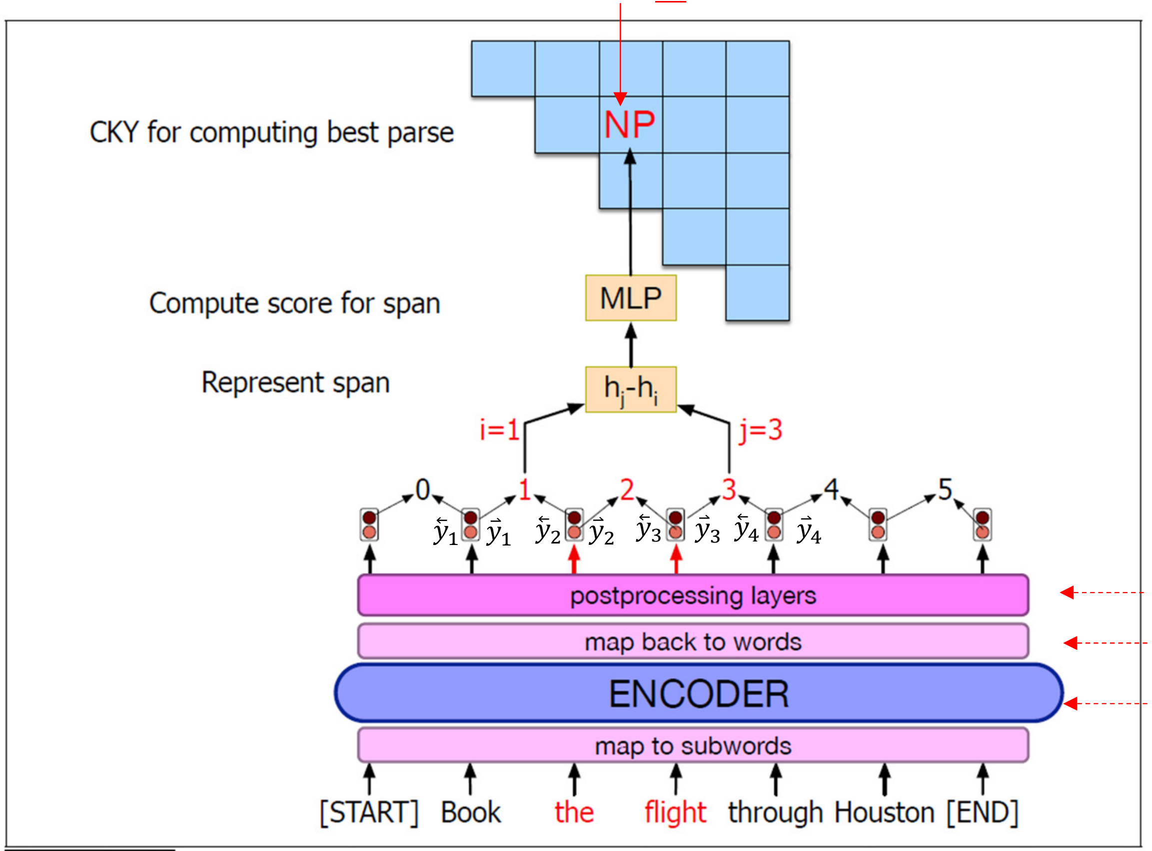


Figure 17.15 A simplified outline of computing the span score for the span *the flight* with the label NP.

Span Score:

$s(i, j, \cdot)$

$= \mathrm{softmax}(\mathrm{M}_2 \mathrm{ReLU}(\mathrm{LayerNorm}(\mathrm{M}_1 \mathbf{v}(i, j) + \mathbf{c}_1)) + \mathbf{c}_2)$

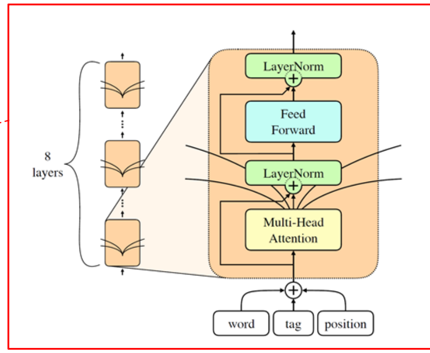
$\mathbf{v}(i, j) = [\overrightarrow{y}_j - \overrightarrow{y}_i ; \overleftarrow{y}_{j+1} - \overleftarrow{y}_{i+1}]$

See next page for more details.

8 Transformer layers

word-level representations

BERT (subwords: word pieces)



Span-Based Neural Constituency Parsing (3/4)

- The output vector of each word y_t is spilt into two halves $(\overleftarrow{y}_t; \overrightarrow{y}_t)$
  - A (leftward-pointing) vector for spans ending at this fencepost (護欄柱), $\overleftarrow{y}_t$, and a (rightward-pointing) vector $\overrightarrow{y}_t$ for spans beginning at this fencepost
  - More specifically, even coordinates contribute to $\overrightarrow{y}_t$ and old coordinates contribute to $\overleftarrow{y}_t$

alternative postprocessing layers

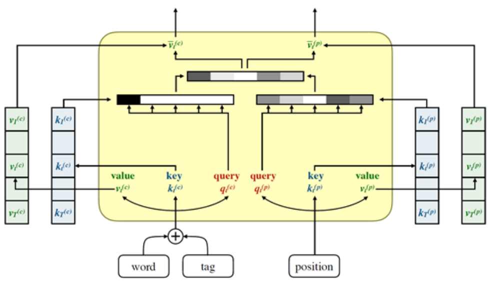
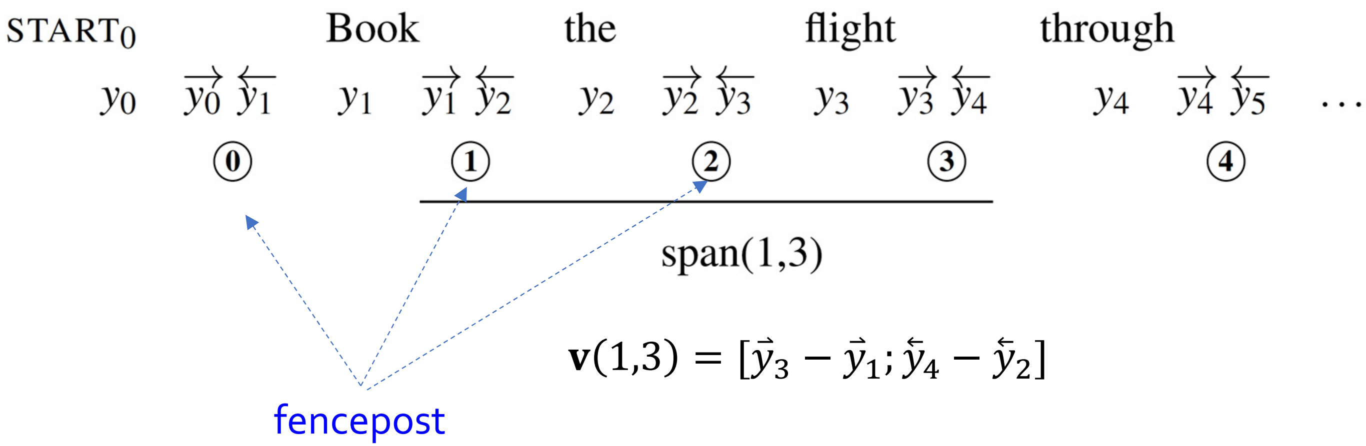


Figure 4: A single attention head, after factoring content and position information. Attention probabilities are calculated separately for the two types of information, and a combined probability distribution is then applied to both types of input information.



Span-Based Neural Constituency Parsing (4/4)

- Choose the final parse tree that has the maximum score

$$\hat{T} = \underset{T}{\operatorname{argmax}}\, s(T)$$

- A variant of the CYK algorithm (Dynamic Programming)
  - For spans of length 1

$$s_{\text{best}} = \max_{l} s(i, i+1, l)$$

  - For other spans (of length >1)

$$s_{\text{best}}(i,j) = \max_{l} s(i,j,l) + \max_{k} [s_{\text{best}}(i,k) + s_{\text{best}}(k,j)]$$

This discriminative parser achieved a state-of-the-art result (with an F1 score of 0.9355) on the Penn Treebank dataset at that time.

The Bottom-Up Chart Parser (1/29)

Kay (1973; 1980)

- A data structure called chart is introduced
  - Allow the parser to store the partial results of matching as it done so far
  - Such that the work would not be reduplicated
- The basic operation of a chart parser involves combining an active arc with a complete constituents (keys)
  - Three kinds of data structures
    - The agenda (to store new complete constituents)
    - The active arcs (i.e. partial parse trees)
    - The chart

- A subtree corresponding to a single grammar rule
- Information about the progress made in completing the subtree
- Position of the subtree has to do with the input

Top-down

NP → ART ADJ N

Bottom-up

Space Complexity vs. Time Complexity

The Bottom-Up Chart Parser (2/29)

- The Bottom-Up Chart Parsing Algorithm

  1. If the agenda is empty, look up the **interpretations** for the next word in the input and add them to the agenda
  2. Select a constituent from the agenda (Call it constituent C from position p_1 to p_2)
  3. For each rule in the grammar of form $X \rightarrow CX_1...X_n$, add **an active arc** of form $X \rightarrow \circ\, CX_1...X_n$ from position p_1 to p_1
  4. Add C to the chart using the following **arc extension algorithm**

     4.1 Insert C **into the chart** from position p_1 to p_2

     4.2 For any active arc of the form $X \rightarrow X_1... \circ C...X_n$ from position p_0 to p_1 add a new active arc $X \rightarrow X_1... C \circ ...X_n$ from p_0 to p_2

     4.3 For any active arc of the form $X \rightarrow X_1... X_n \circ C$ from position p_0 to p_1, then add a new constituent of type X from p_0 to p_2 **to the agenda**

Loop until
no input left

The Bottom-Up Chart Parser (3/29)

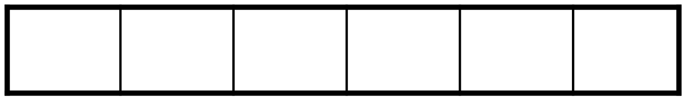
- Example

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
holds: N, V
Water: N

Initialization

Chart:



Input: $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

Note that depth-first strategy is used here
=> The agenda is implemented as a stack.

The Bottom-Up Chart Parser (4/29)

- The Bottom-Up Chart Parsing Algorithm
  1. If the agenda is empty, look up the **interpretations** for the next word in the input and add them to the agenda
  2. Select a constituent from the agenda (Call it constituent C from position p_1 to p_2)
  3. For each rule in the grammar of form $X \rightarrow CX_1...X_n$, add **an active arc** of form $X \rightarrow \circ CX_1...X_n$ from position p_1 to p_1
  4. Add C to the chart using the following **arc extension algorithm**
     - 4.1 Insert C **into the chart** from position p_1 to p_2
     - 4.2 For any active arc of the form $X \rightarrow X_1... \circ C...X_n$ from position p_0 to p_1 add a new active arc $X \rightarrow X_1... C \circ ...X_n$ from p_0 to p_2
     - 4.3 For any active arc of the form $X \rightarrow X_1... X_n \circ C$ from position p_0 to p_1, then add a new constituent of type X from p_0 to p_2 **to the agenda**

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 1

Enter ART1: (*the* from 1 to 2) Look at next word

Chart:

Agenda:

ART1 1,2

NP → ∘ ART ADJ N

NP → ∘ ART N

The Bottom-Up Chart Parser (5/29)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

- The Bottom-Up Chart Parsing Algorithm
  1. If the agenda is empty, look up the **interpretations** for the next word in the input and add them to the agenda
  2. Select a constituent from the agenda (Call it constituent C from position p_1 to p_2)
  3. For each rule in the grammar of form $X \rightarrow CX_1...X_n$, add **an active arc** of form $X \rightarrow \circ CX_1...X_n$ from position p_1 to p_1
  4. Add C to the chart using the following **arc extension algorithm**
     4.1 Insert C **into the chart** from position p_1 to p_2
     4.2 For any active arc of the form $X \rightarrow X_1... \circ C...X_n$ from position p_0 to p_1 add a new active arc $X \rightarrow X_1... C \circ ...X_n$ from p_0 to p_2
     4.3 For any active arc of the form $X \rightarrow X_1... X_n \circ C$ from position p_0 to p_1, then add a new constituent of type X from p_0 to p_2 **to the agenda**

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 1

Enter ART1: (*the* from 1 to 2) (using the arc extension algorithm)

Chart:

ART1					

Agenda:

~~ART1 1,2~~

NP → ART ∘ ADJ N

NP → ART ∘ N

Add active arcs

The Bottom-Up Chart Parser (6/29)

- The Bottom-Up Chart Parsing Algorithm
  1. If the agenda is empty, look up the **interpretations** for the next word in the input and add them to the agenda
  2. Select a constituent from the agenda (Call it constituent C from position p_1 to p_2)
  3. For each rule in the grammar of form $X \rightarrow CX_1 \ldots X_n$, add **an active arc** of form $X \rightarrow \circ\, CX_1 \ldots X_n$ from position p_1 to p_1
  4. Add C to the chart using the following **arc extension algorithm**
     - 4.1 Insert C **into the chart** from position p_1 to p_2
     - 4.2 For any active arc of the form $X \rightarrow X_1 \ldots \circ\, C \ldots X_n$ from position p_0 to p_1 add a new active arc $X \rightarrow X_1 \ldots C \circ \ldots X_n$ from p_0 to p_2
     - 4.3 For any active arc of the form $X \rightarrow X_1 \ldots X_n \circ C$ from position p_0 to p_1, then add a new constituent of type X from p_0 to p_2 **to the agenda**

- Example $\quad {}_1$ The ${}_2$ large ${}_3$ can ${}_4$ holds ${}_5$ the ${}_6$ water

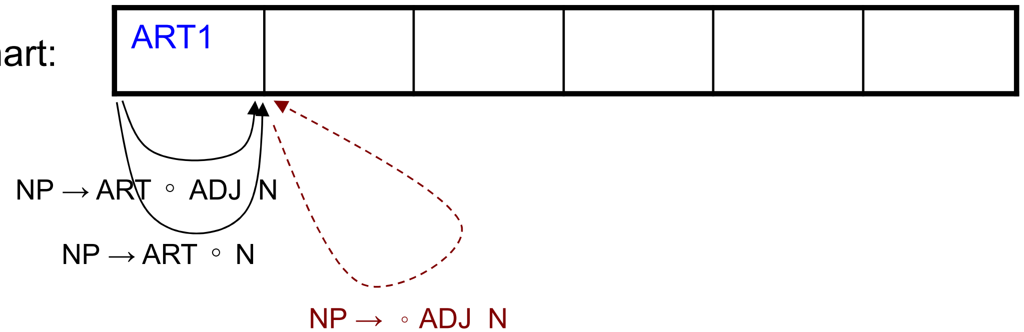
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 2 Look at next word

Enter ADJ1: ("*large*" from 2 to 3)

Chart:



Agenda:

ADJ1 2,3

The Bottom-Up Chart Parser (7/29)

- The Bottom-Up Chart Parsing Algorithm
  1. If the agenda is empty, look up the **interpretations** for the next word in the input and add them to the agenda
  2. Select a constituent from the agenda (Call it constituent C from position p_1 to p_2)
  3. For each rule in the grammar of form $X \rightarrow CX_1 \ldots X_n$, add **an active arc** of form $X \rightarrow \circ\, CX_1 \ldots X_n$ from position p_1 to p_1
  4. Add C to the chart using the following **arc extension algorithm**
     - 4.1 Insert C **into the chart** from position p_1 to p_2
     - 4.2 For any active arc of the form $X \rightarrow X_1 \ldots \circ\, C \ldots X_n$ from position p_0 to p_1 add a new active arc $X \rightarrow X_1 \ldots C \circ \ldots X_n$ from p_0 to p_2
     - 4.3 For any active arc of the form $X \rightarrow X_1 \ldots X_n \circ C$ from position p_0 to p_1, then add a new constituent of type X from p_0 to p_2 **to the agenda**

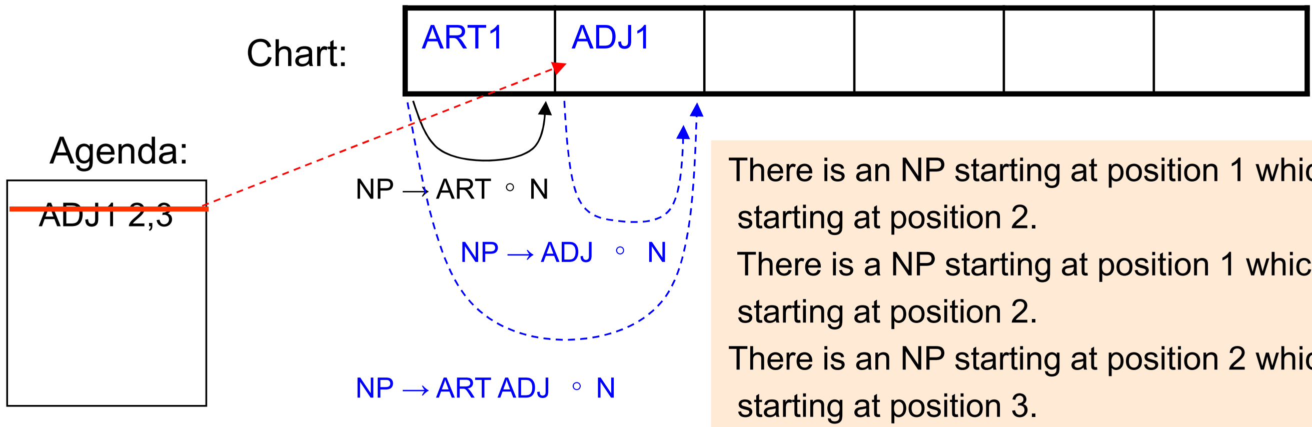
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 2 (using the arc extension algorithm)

Enter ADJ1: ("*large*" from 2 to 3)



There is an NP starting at position 1 which needs an N starting at position 2.
There is a NP starting at position 1 which needs an ADJ starting at position 2.
There is an NP starting at position 2 which needs an N starting at position 3.

The Bottom-Up Chart Parser (8/29)

- The Bottom-Up Chart Parsing Algorithm
  1. If the agenda is empty, look up the **interpretations** for the next word in the input and add them to the agenda
  2. Select a constituent from the agenda (Call it constituent C from position p_1 to p_2)
  3. For each rule in the grammar of form $X \rightarrow CX_1...X_n$, add **an active arc** of form $X \rightarrow \circ\ CX_1...X_n$ from position p_1 to p_1
  4. Add C to the chart using the following **arc extension algorithm**
     - 4.1 Insert C **into the chart** from position p_1 to p_2
     - 4.2 For any active arc of the form $X \rightarrow X_1... \circ C...X_n$ from position p_0 to p_1 add a new active arc $X \rightarrow X_1... C \circ ...X_n$ from p_0 to p_2
     - 4.3 For any active arc of the form $X \rightarrow X_1... X_n \circ C$ from position p_0 to p_1, then add a new constituent of type X from p_0 to p_2 **to the agenda**

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

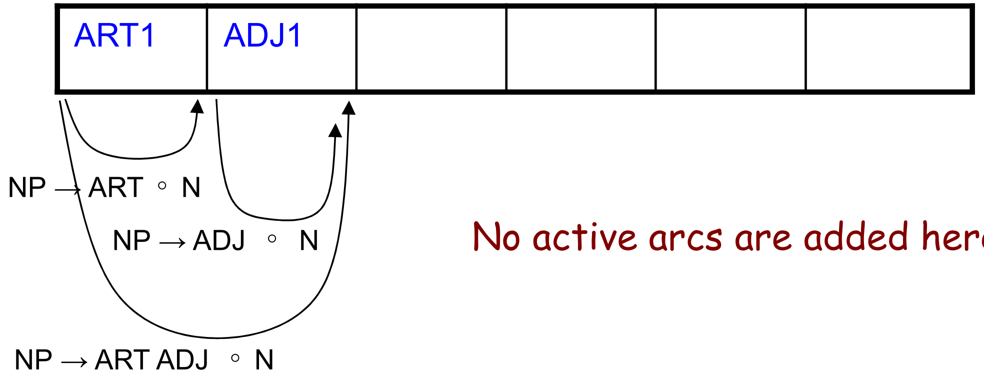
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 3 Look at next word

Enter N1: (*"can"* from 3 to 4)

Chart:



Agenda:

N1 3,4
AUX1 3,4

No active arcs are added here!

The Bottom-Up Chart Parser (9/29)

- Example

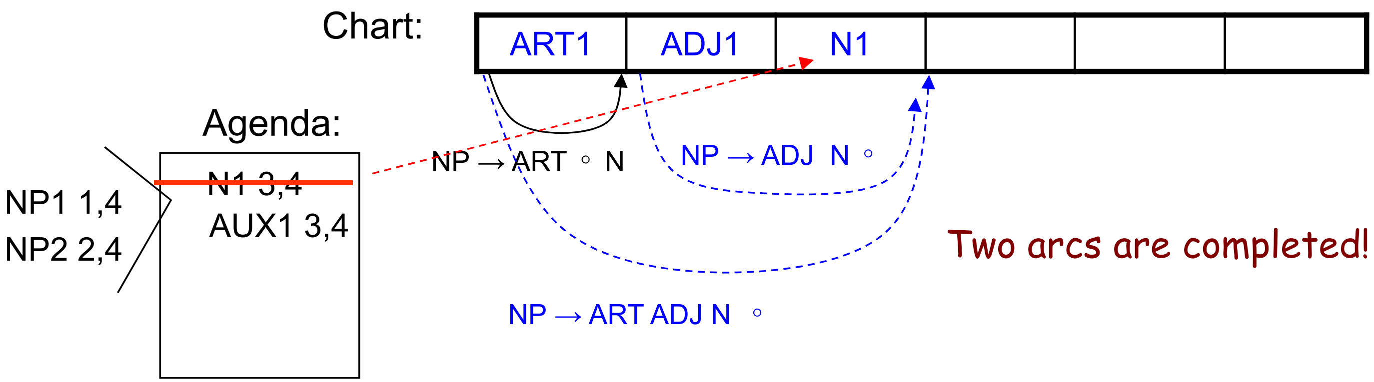
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 3

Enter N1: (*"can"* from 3 to 4) (using the arc extension algorithm)



The Bottom-Up Chart Parser (10/29)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

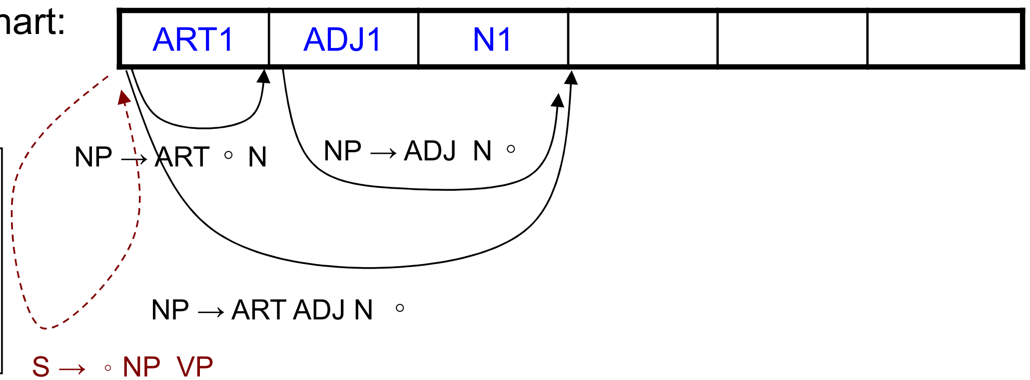
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 4

Enter NP1: ("*the* large can" from 1 to 4)

Chart:



Agenda:

NP1 1,4
NP2 2,4
AUX1 3,4

The Bottom-Up Chart Parser (11/29)

- Example

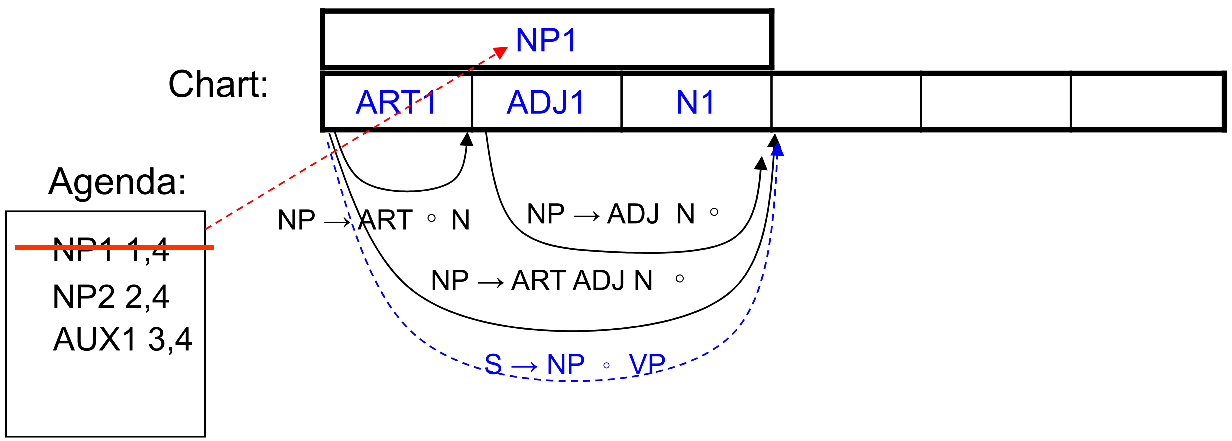
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 4

Enter NP1: ("*the* large can" from 1 to 4) (using the arc extension algorithm)



The Bottom-Up Chart Parser (12/29)

- Example

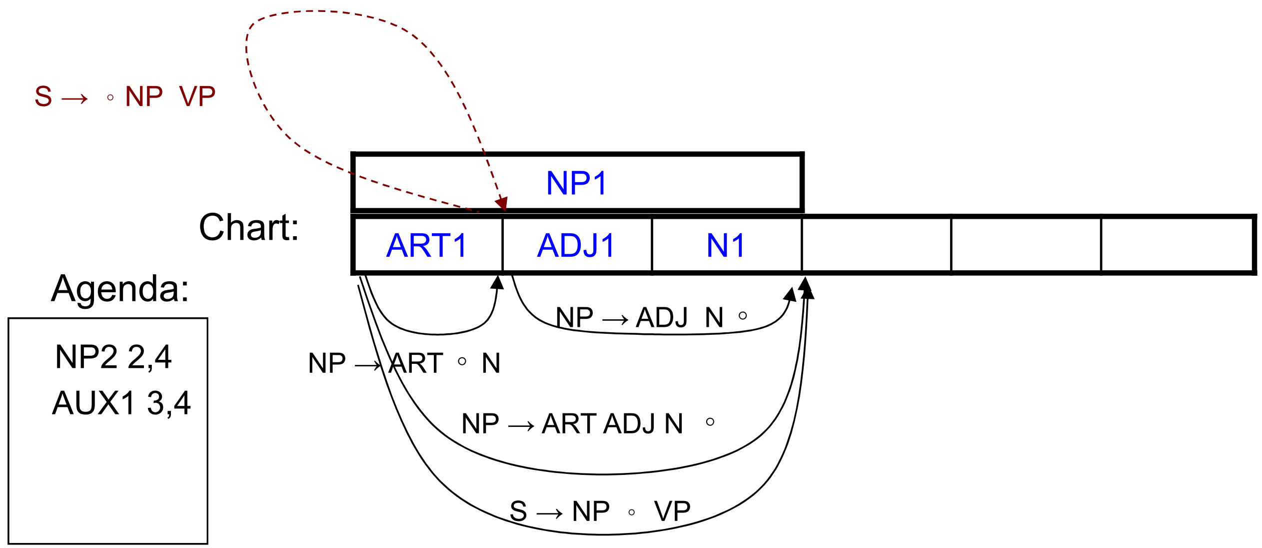
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 5

Enter NP2: (*"large can"* from 2 to 4)



The Bottom-Up Chart Parser (13/29)

- Example

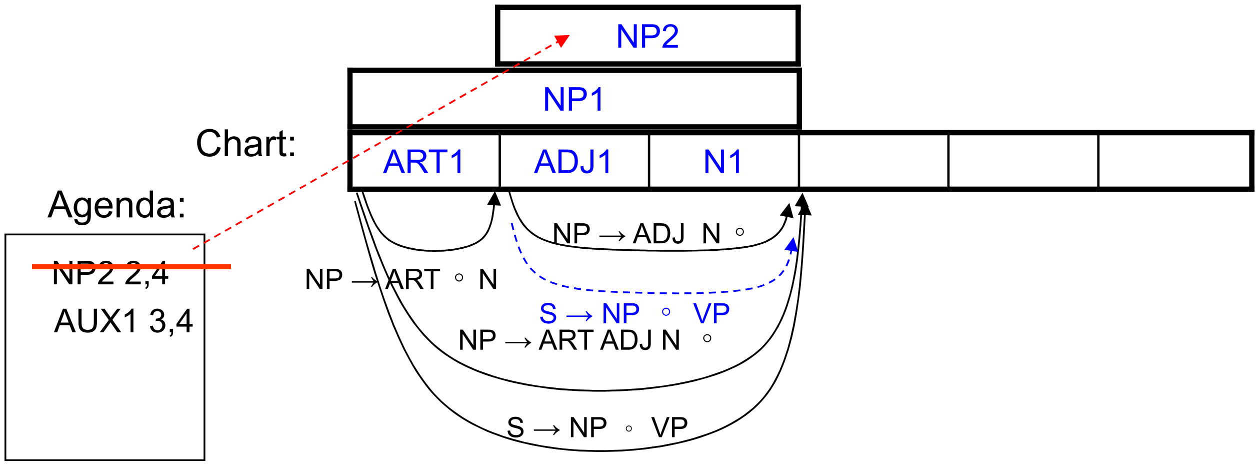
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 5 (using the arc extension algorithm)

Enter NP2: (*“large can”* from 2 to 4)



The Bottom-Up Chart Parser (14/29)

- Example

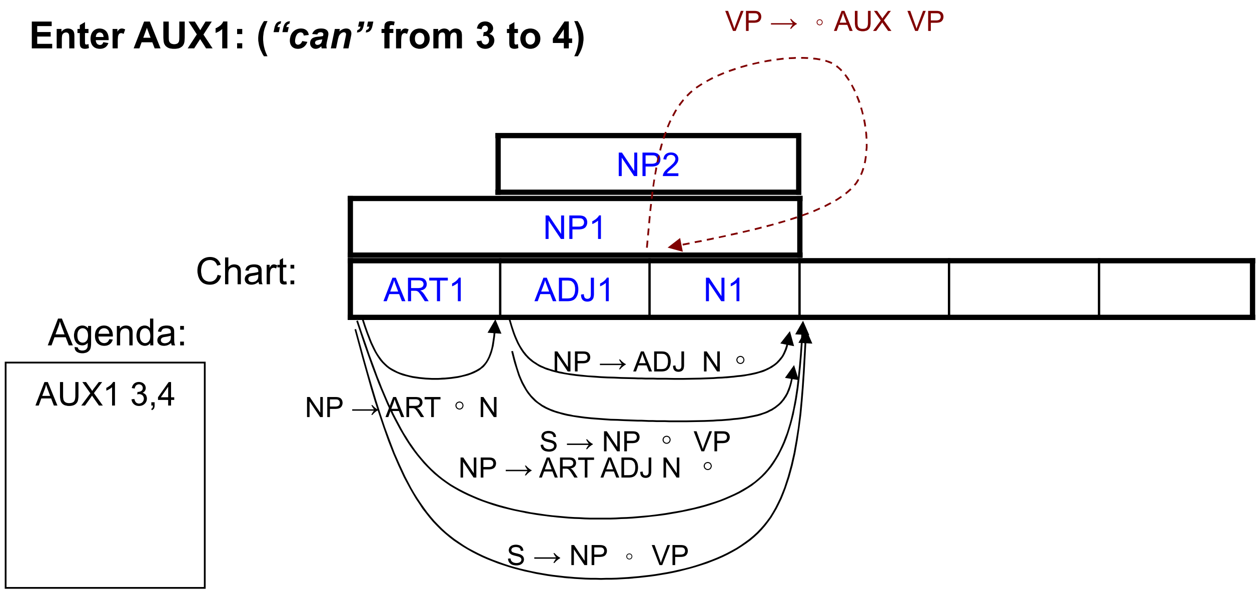
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 6

Enter AUX1: (*"can"* from 3 to 4)



The Bottom-Up Chart Parser (15/29)

- Example

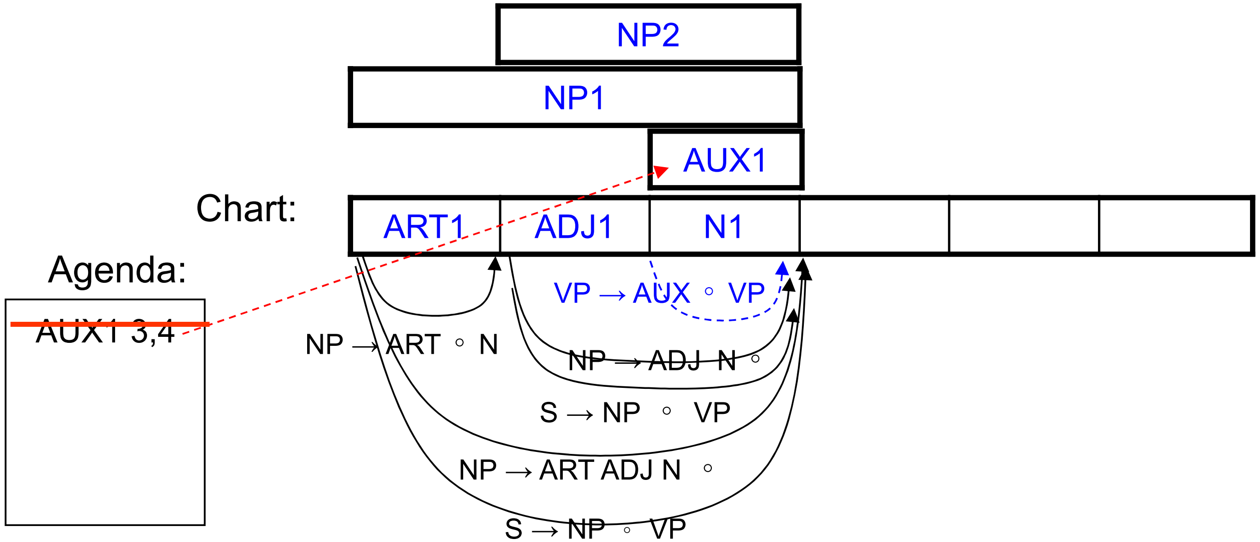
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 6 (using the arc extension algorithm)

Enter AUX1: (*“can”* from 3 to 4)



The Bottom-Up Chart Parser (16/29)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

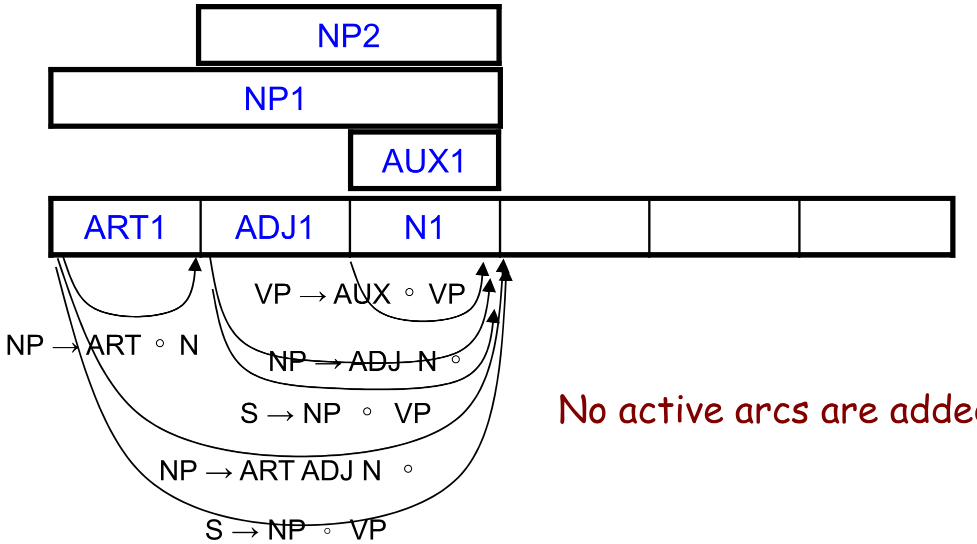
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 7 Look at next word

Enter N2: (*“hold”* from 4 to 5)

Chart:



Agenda:

N2 4,5
V1 4,5

No active arcs are added here!

The Bottom-Up Chart Parser (17/29)

- Example

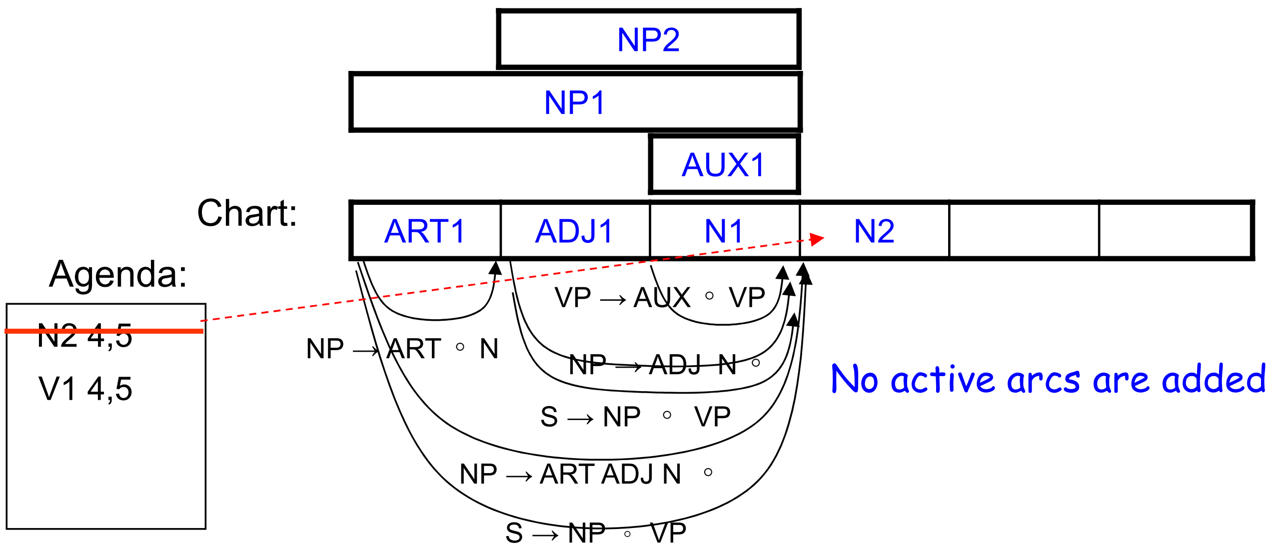
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 7 (using the arc extension algorithm)

Enter N2: (*"hold"* from 4 to 5)



No active arcs are added here!

The Bottom-Up Chart Parser (18/29)

- Example

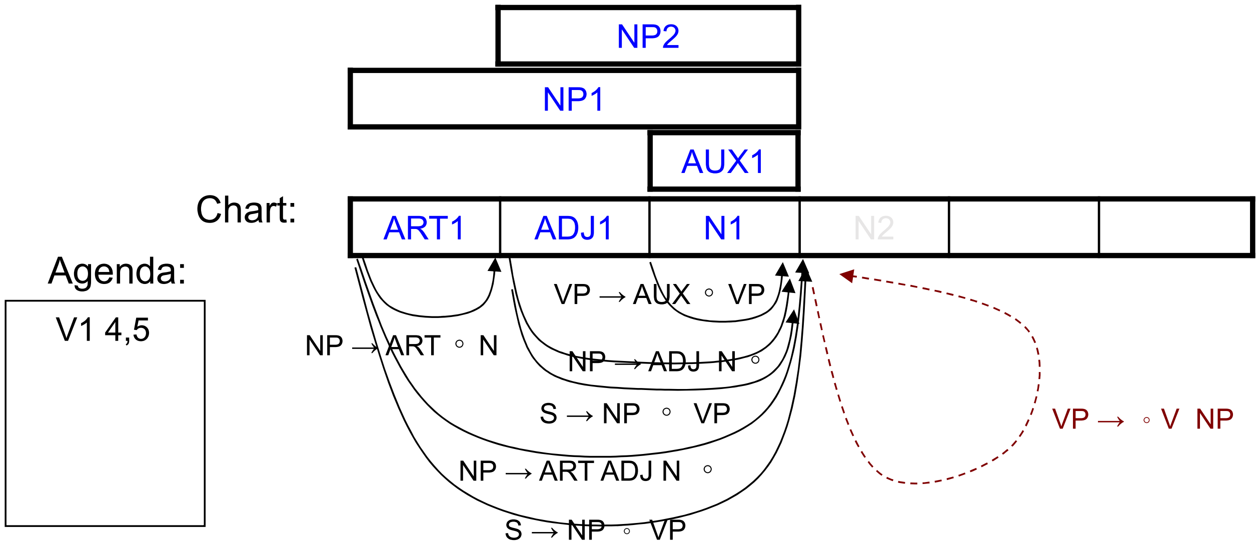
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 8

Enter V1: (*"hold"* from 4 to 5)



The Bottom-Up Chart Parser (19/29)

- Example

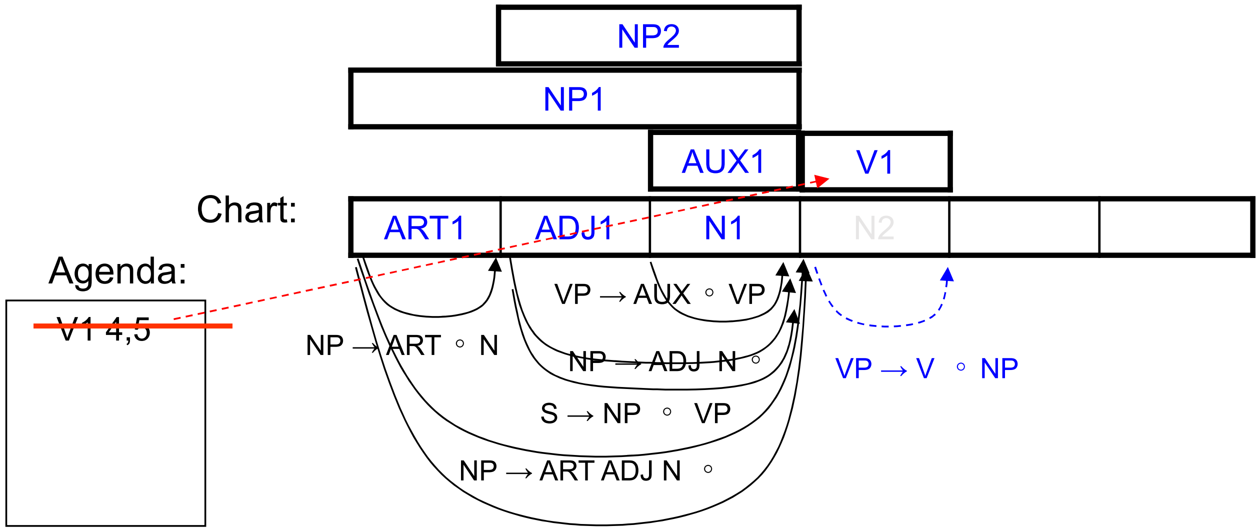
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 8 (using the arc extension algorithm)

Enter V1: (*"hold"* from 4 to 5)



The Bottom-Up Chart Parser (20/29)

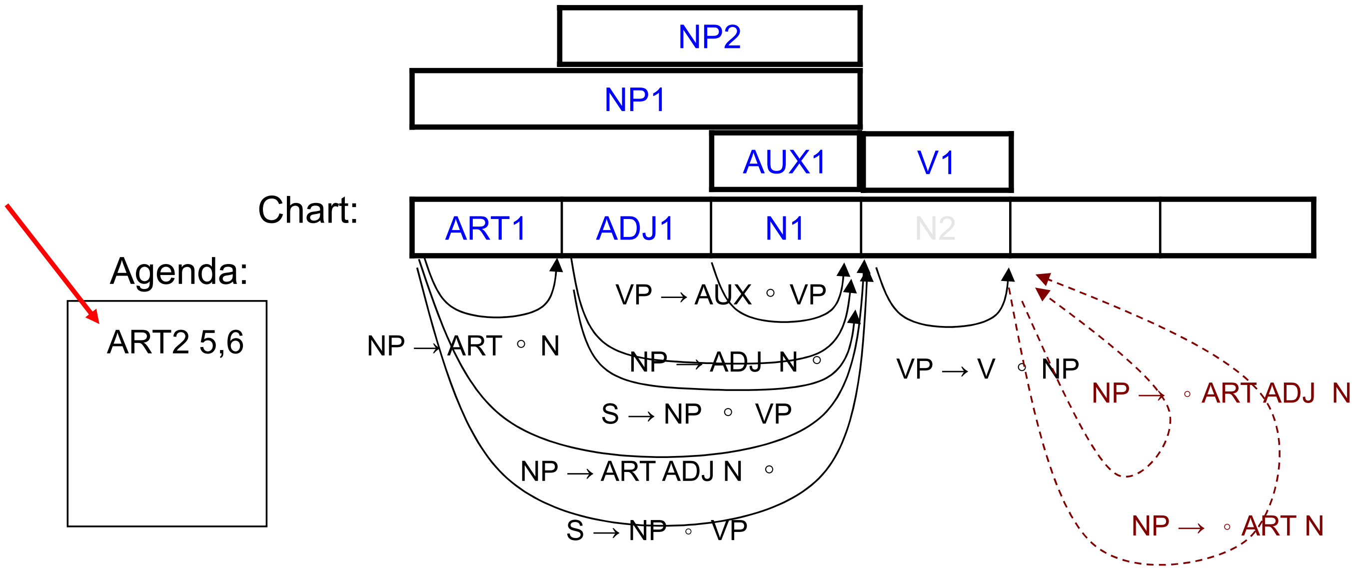
- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 9 Look at next word
Enter ART2: ("*the*" from 5 to 6)



The Bottom-Up Chart Parser (21/29)

- Example

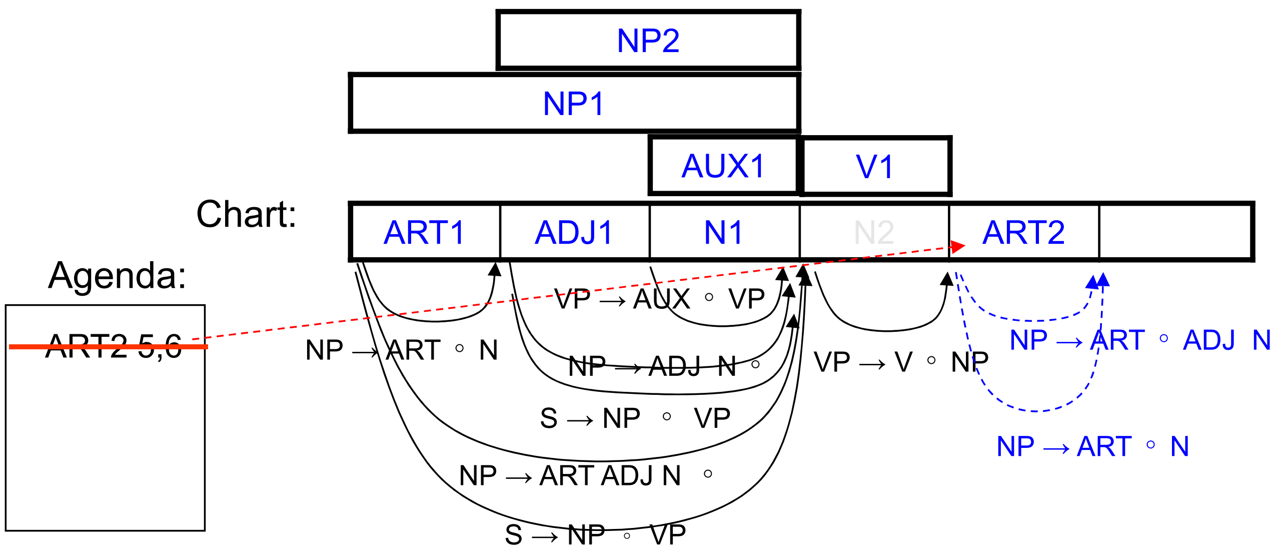
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 9 (using the arc extension algorithm)

Enter ART2: ("*the*" from 5 to 6)



The Bottom-Up Chart Parser (22/29)

- Example

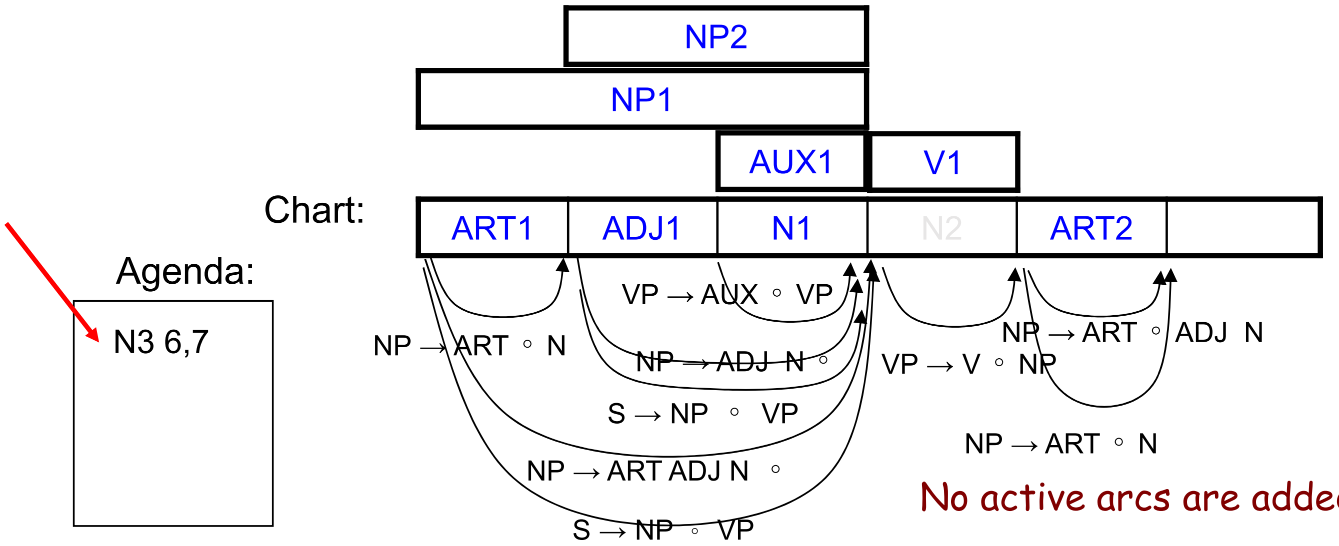
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 10 Look at next word

Enter N3: (*"water"* from 6 to 7)



No active arcs are added here!

The Bottom-Up Chart Parser (23/29)

- Example

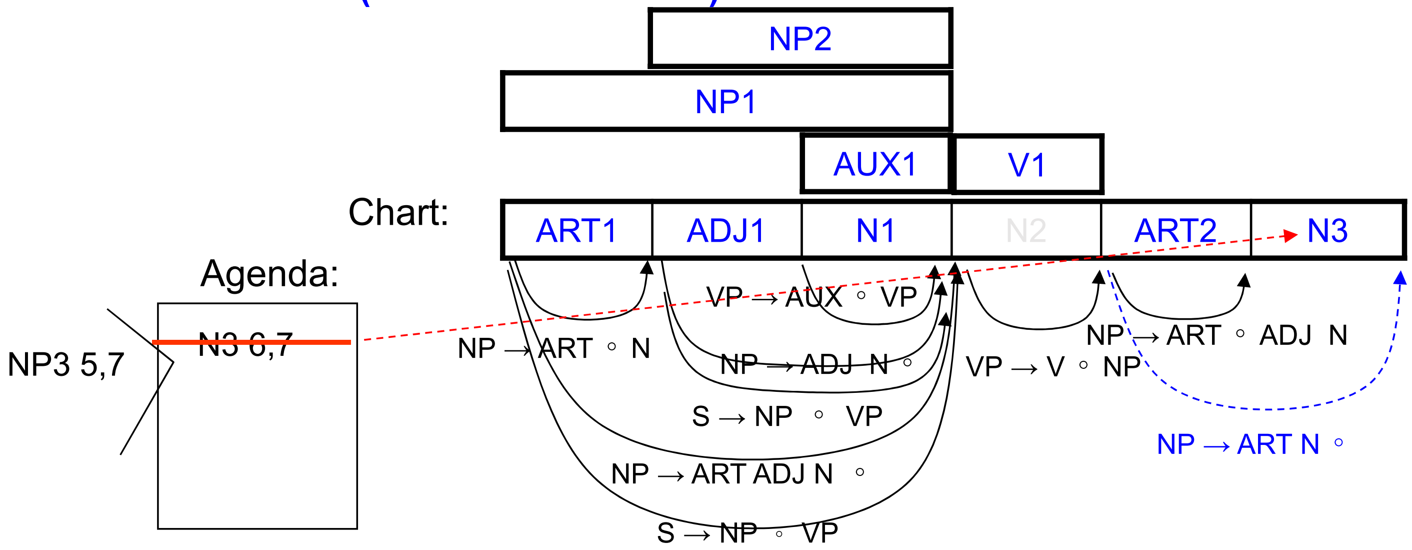
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 10 (using the arc extension algorithm)

Enter N3: (“*water*” from 6 to 7)



The Bottom-Up Chart Parser (24/29)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

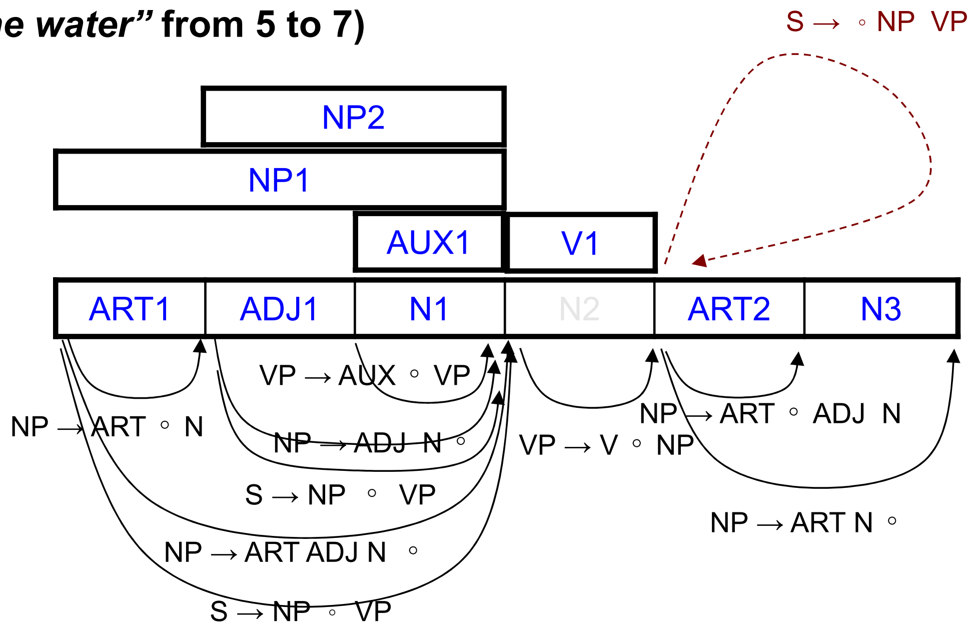
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 11

Enter NP3: ("*the water*" from 5 to 7)

Chart:



Agenda:

NP3 5,7

The Bottom-Up Chart Parser (25/29)

- Example

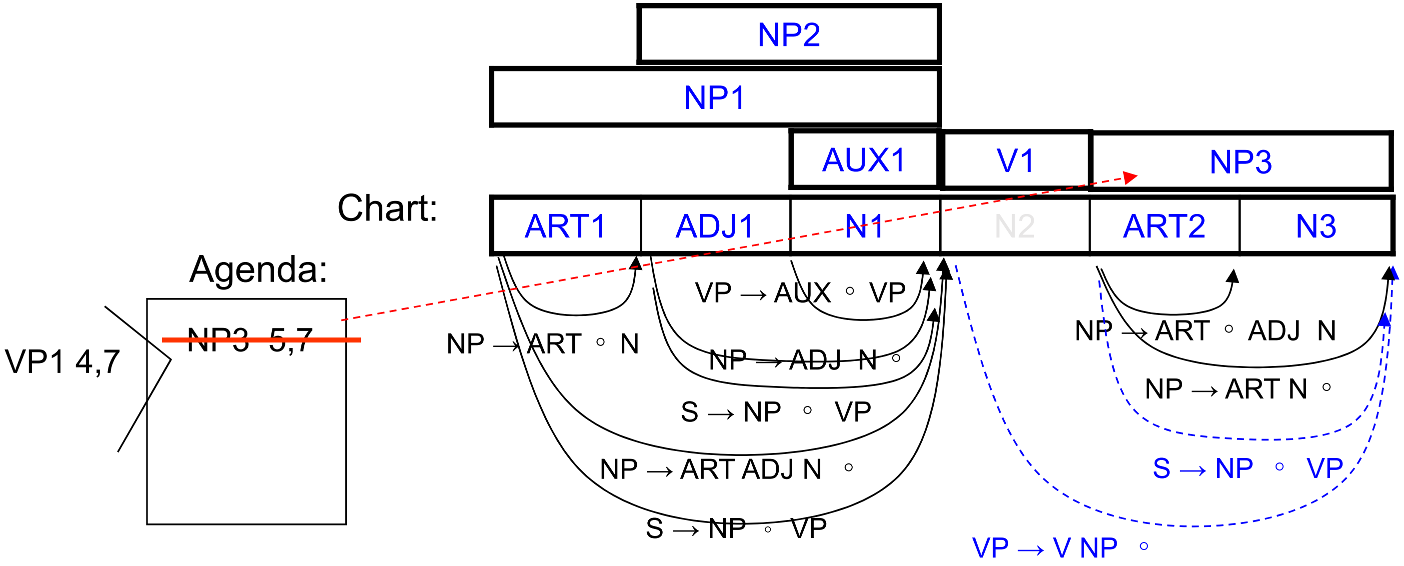
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 11 (using the arc extension algorithm)

Enter NP3: (*"the water"* from 5 to 7)



The Bottom-Up Chart Parser (26/29)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

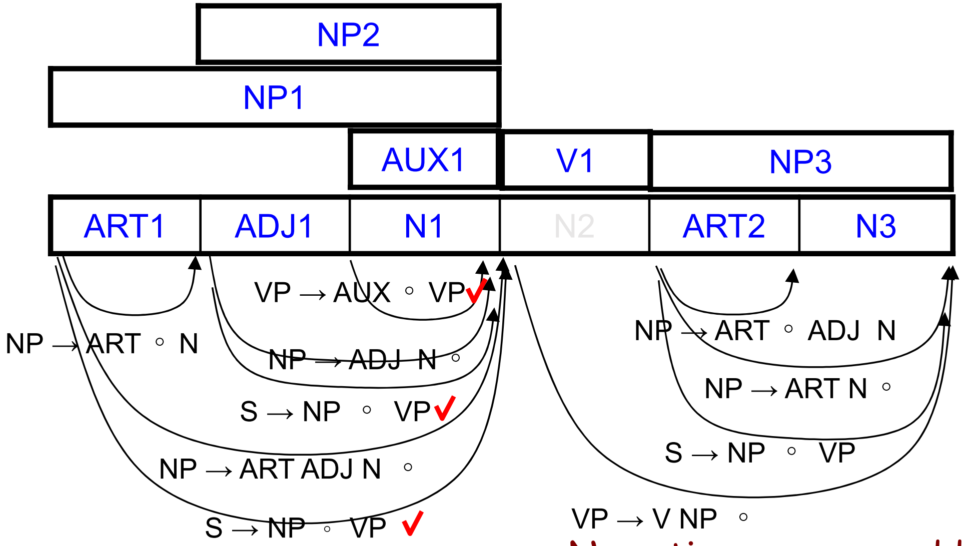
Loop 12

Enter VP1: (*"hold the water"* from 4 to 7)

Agenda:

VP1 4,7

Chart:



No active arcs are added here!

The Bottom-Up Chart Parser (27/29)

- Example

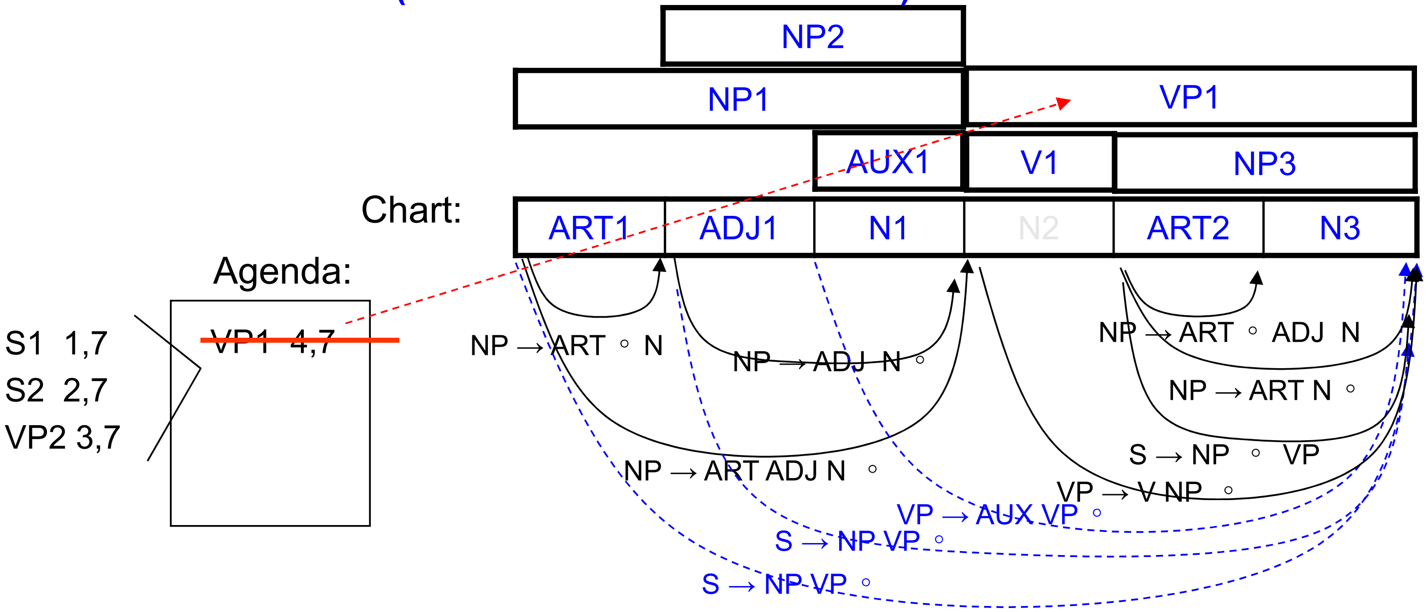
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 12 (using the arc extension algorithm)

Enter VP1: (*"hold the water"* from 4 to 7)



The Bottom-Up Chart Parser (28/29)

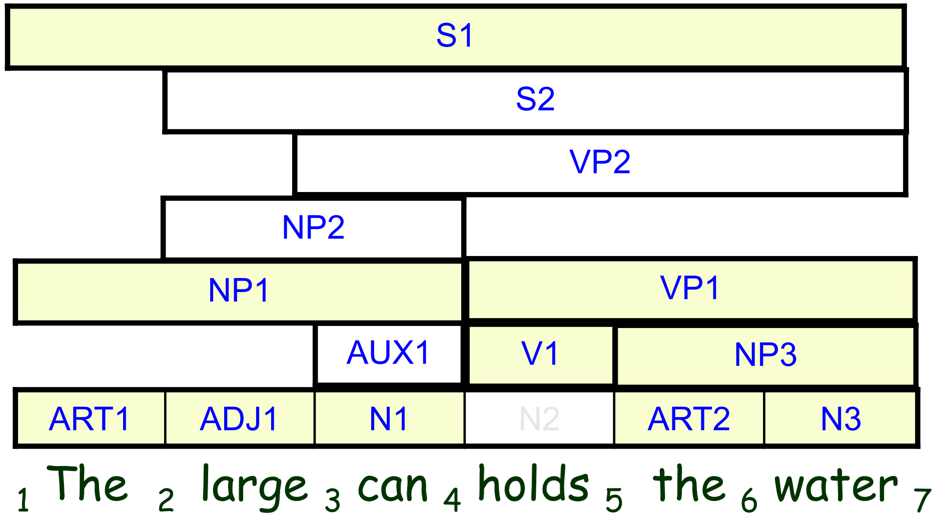
- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

The final Chart



Since you have derived an S covering the entire sentence, you can stop successfully. If you wanted to find all possible interpretations for the sentence, you would continue parsing until the agenda became empty.

The Bottom-Up Chart Parser (29/29)

- Characteristics
  - The algorithm always moves forward through the chart *making additions* as it goes
  - Arcs are never removed and the algorithm never backtracks to a previous chart entry once it has moved on
  - Different *S* structures might share the common subparts represented in the chart only once

78

The Top-Down Chart Parser (1/27)

- The Top-Down Chart Parsing Algorithm

Initialization: For every rule in the grammar of form $S \rightarrow X_1 ... X_k$, add an arc labeled $S \rightarrow \circ X_1 ... X_k$ using **the arc introduction algorithm**

1. If the agenda is empty, look up the interpretations of the next word and add them to the agenda
2. Select a constituent C from the agenda
3. Using **the arc extension algorithm**, combine C with every active arc on the chart. Any new constituents are added to the agenda
4. For any active arcs created in step 3, add them to the chart using the following **top-down arc introduction algorithm**
   - To add an arc $S \rightarrow C_1 ... \circ C_i ... C_n$ ending at position j, do the following:

     For each rule in the grammar of form $C_i \rightarrow X_1 ... X_k$, recursively add the new arc $C_i \rightarrow \circ X_1 ... X_k$ from position j to j

Loop until no input left

The Top-Down Chart Parser (2/27)

- Recall "**the arc extension algorithm**"
  - Insert C **into the chart** from position p_1 to p_2
  - For any active arc of the form $X \rightarrow X_1 ... \circ C ... X_n$ from position p_0 to p_1 add a new active arc $X \rightarrow X_1 ... C \circ ... X_n$ from p_0 to p_2
  - For any active arc of the form $X \rightarrow X_1 ... X_n \circ C$ from position p_0 to p_1, then add a new constituent of type X from p_0 to p_2 **to the agenda**

The Top-Down Chart Parser (3/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Initialization: (using the arc introduction algorithm)

Chart:

Agenda:

Note that no checking of 3rd-person-sg or non-3rd-person-sg verbs was applied here.

S → ◦ NP VP
NP → ◦ ART ADJ N
NP → ◦ ART N
NP → ◦ ADJ N

using the arc introduction algorithm

The Top-Down Chart Parser (4/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 1

Enter ART1 (*"the"* from 1 to 2): Look at next word

Chart:

Agenda:

ART1 1,2

S → ◦ NP VP
NP → ◦ ART ADJ N
NP → ◦ ART N
NP → ◦ ADJ N

The Top-Down Chart Parser (5/27)

- Example

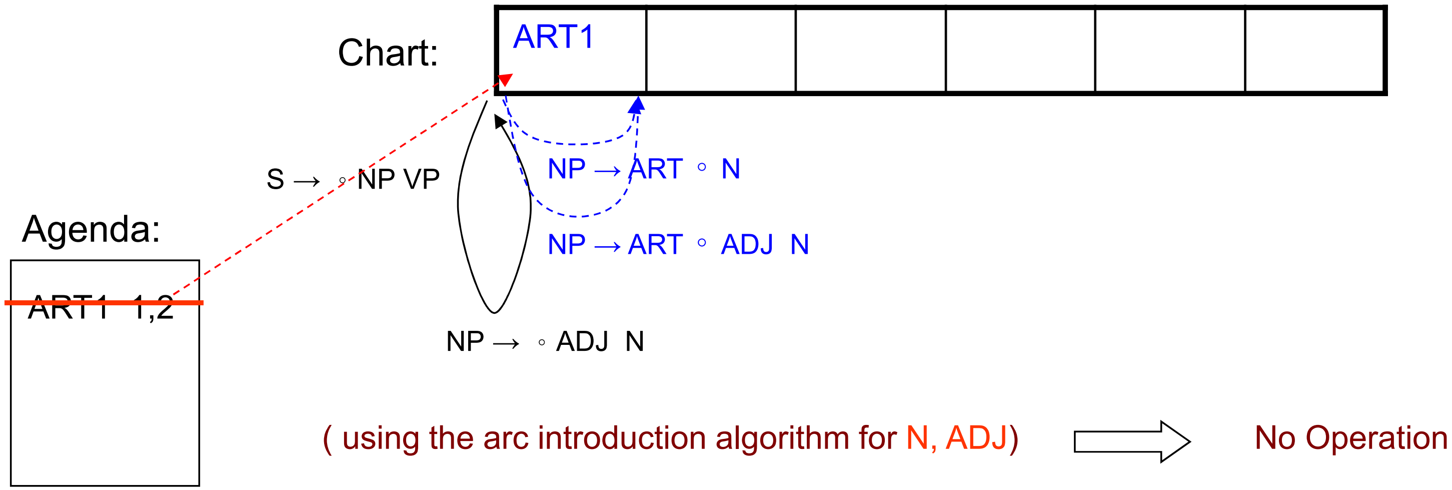
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 1

Enter ART1 (*"the"* from 1 to 2): (using the arc extension algorithm)



The Top-Down Chart Parser (6/27)

- Example

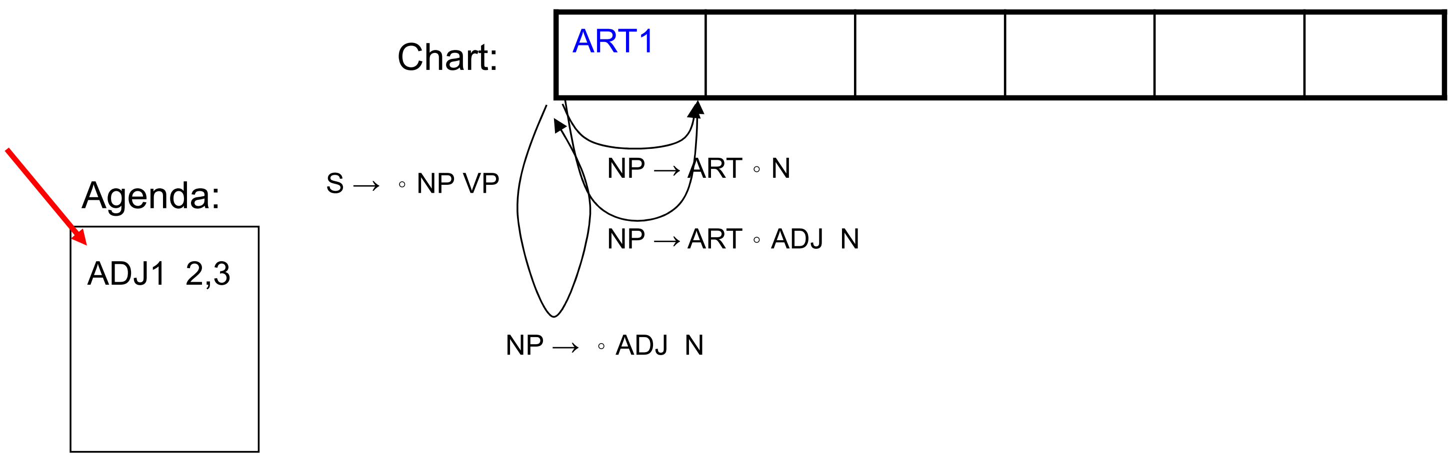
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 2

Enter ART1 (*"large"* from 2 to 3): Look at next word



The Top-Down Chart Parser (7/27)

- Example

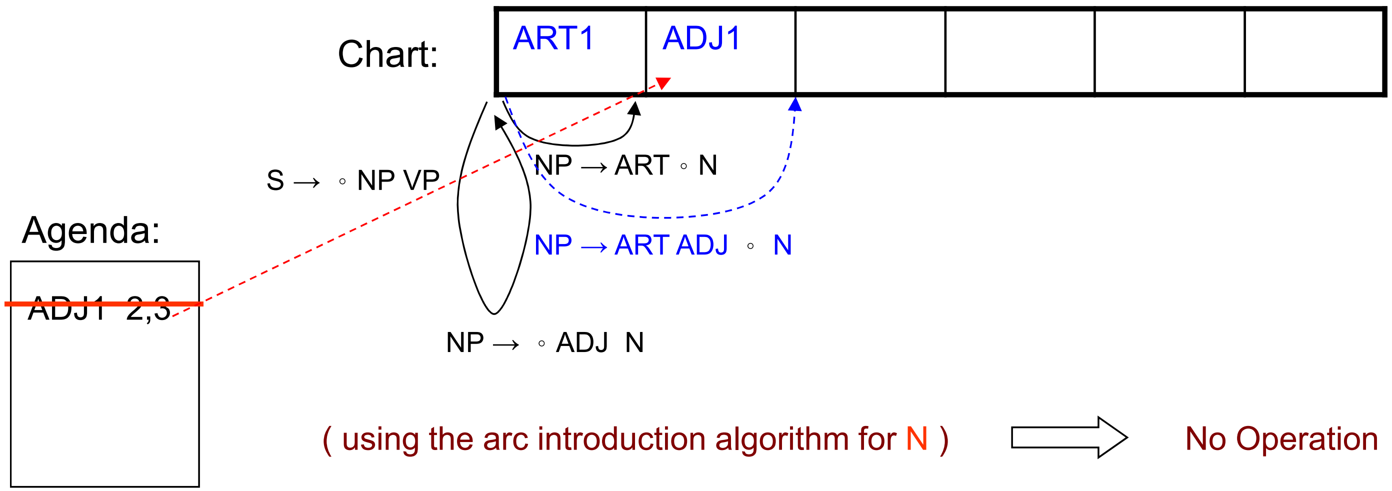
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 2

Enter ADJ1 (“*large*” from 2 to 3): (using the arc extension algorithm)



(using the arc introduction algorithm for N) ⟹ No Operation

The Top-Down Chart Parser (8/27)

- Example

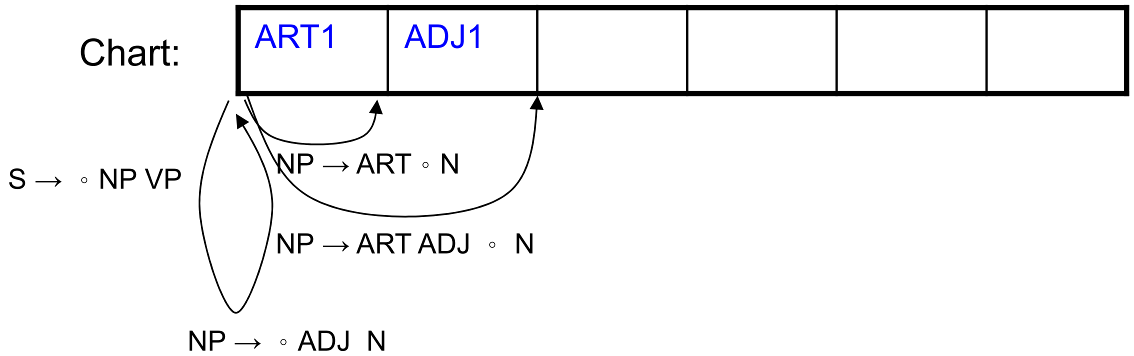
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 3

Enter N1 (*"can"* from 3 to 4): Look at next word



Agenda:

N1 3,4
AUX1 3,4

The Top-Down Chart Parser (9/27)

- Example

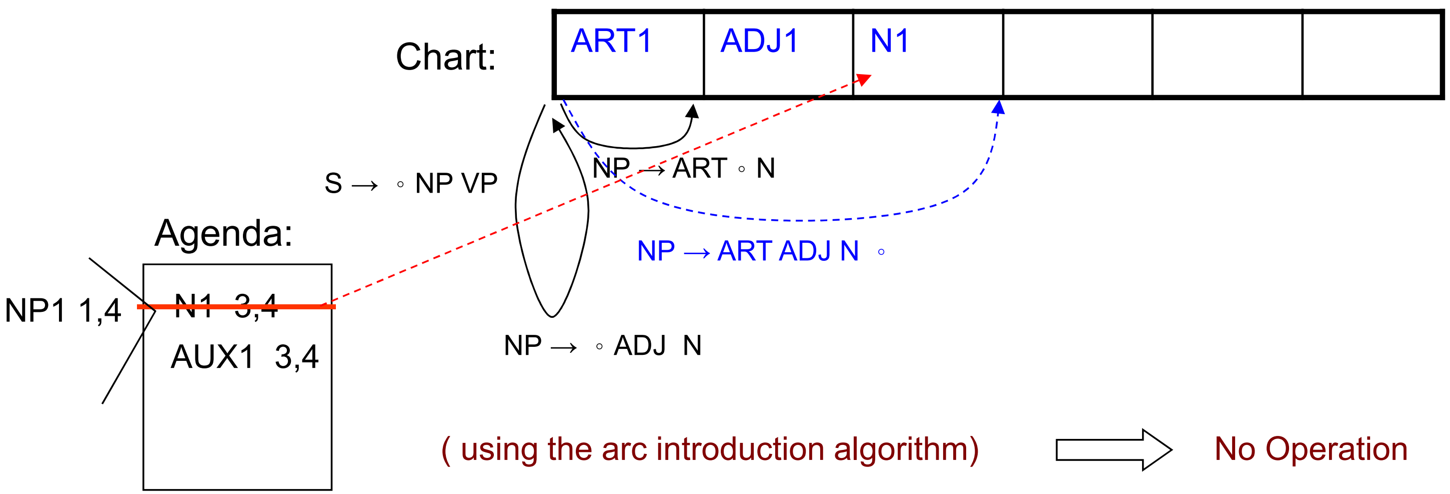
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 3

Enter N1 (“*can*” from 3 to 4): (using the arc extension algorithm)



(using the arc introduction algorithm) ⇨ No Operation

The Top-Down Chart Parser (10/27)

- Example

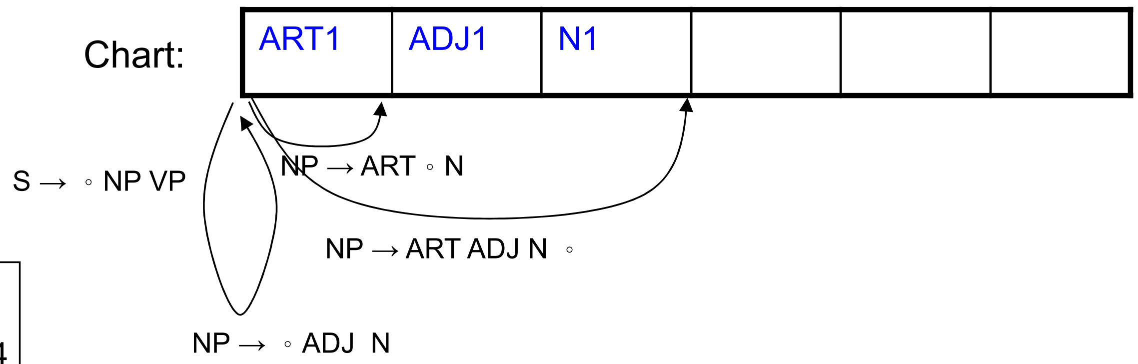
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 4

Enter NP1 (*"the large can"* from 1 to 4):



Agenda:

NP1 1,4
AUX1 3,4

The Top-Down Chart Parser (11/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

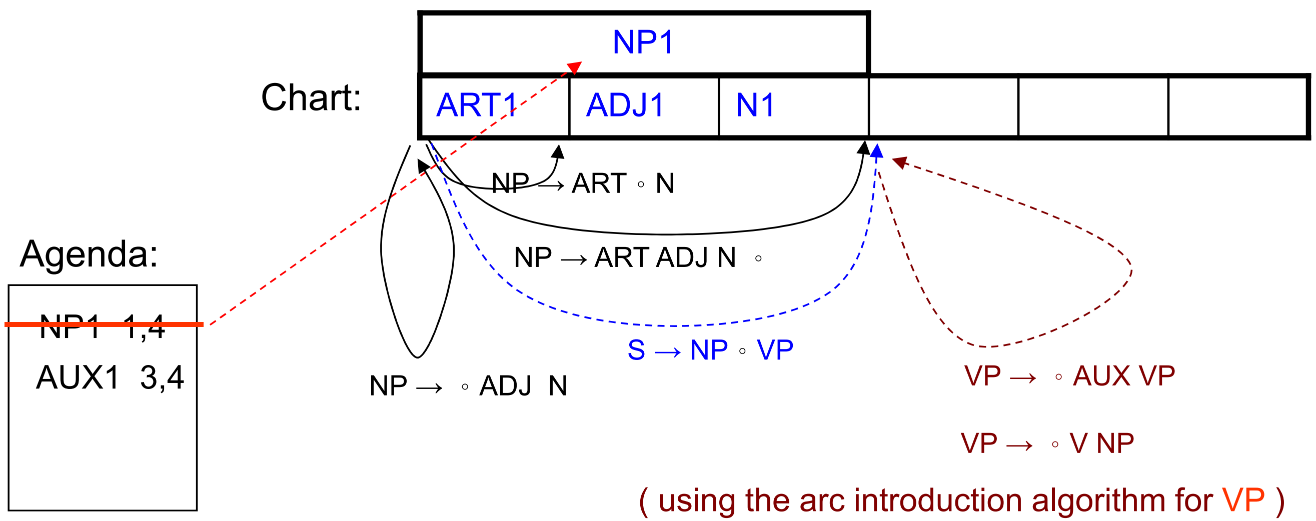
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 4

Enter NP1 (*"the large can"* from 1 to 4):

(using the arc extension algorithm)



(using the arc introduction algorithm for VP)

The Top-Down Chart Parser (12/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

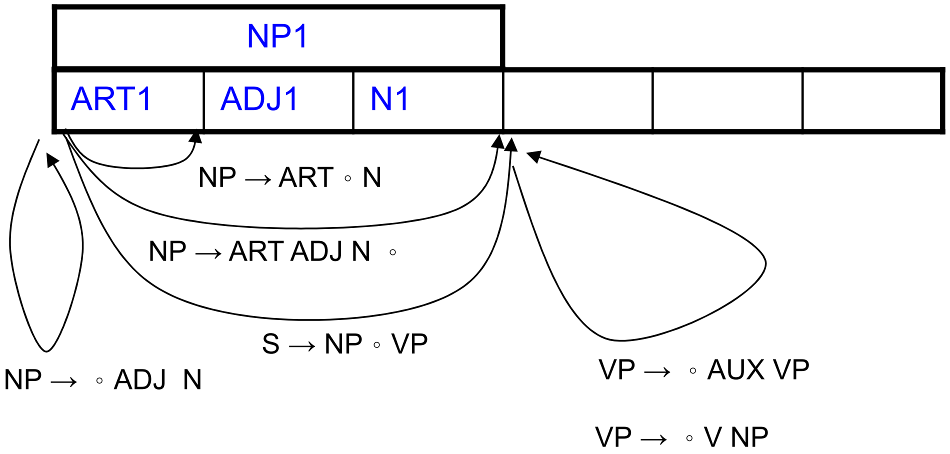
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 5

Enter AUX1 (*"can"* from 3 to 4):

Chart:



Agenda:

AUX1 3,4

The Top-Down Chart Parser (13/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

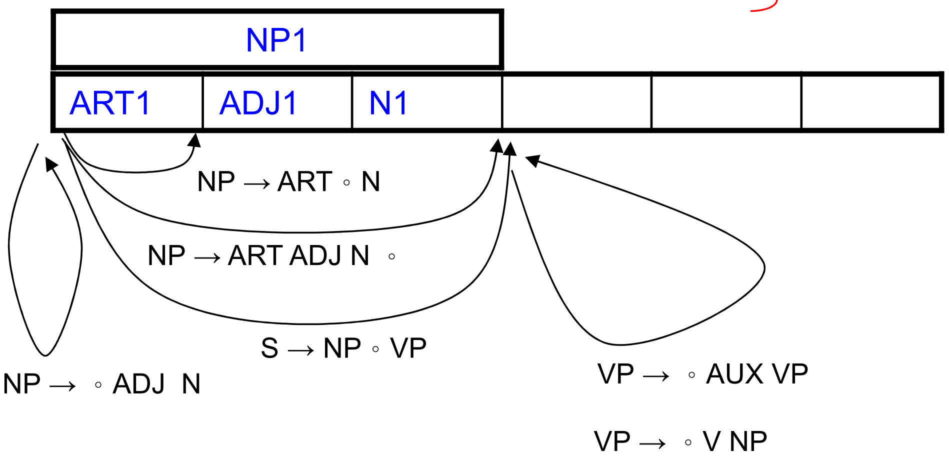
Loop 5

Enter AUX1 (*"can"* from 3 to 4):

(using the arc extension algorithm)
(using the arc introduction algorithm)

No Operation

Chart:



Agenda:

~~AUX1 3,4~~

The Top-Down Chart Parser (14/27)

- Example

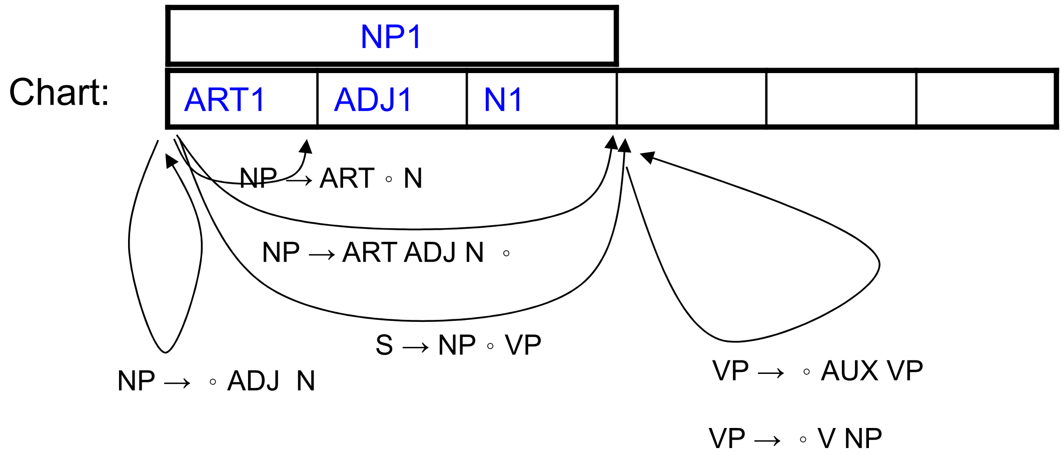
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 6

Enter N2 (*"holds"* from 4 to 5): Look at next word



Agenda:

N2 4,5
V1 4,5

The Top-Down Chart Parser (15/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

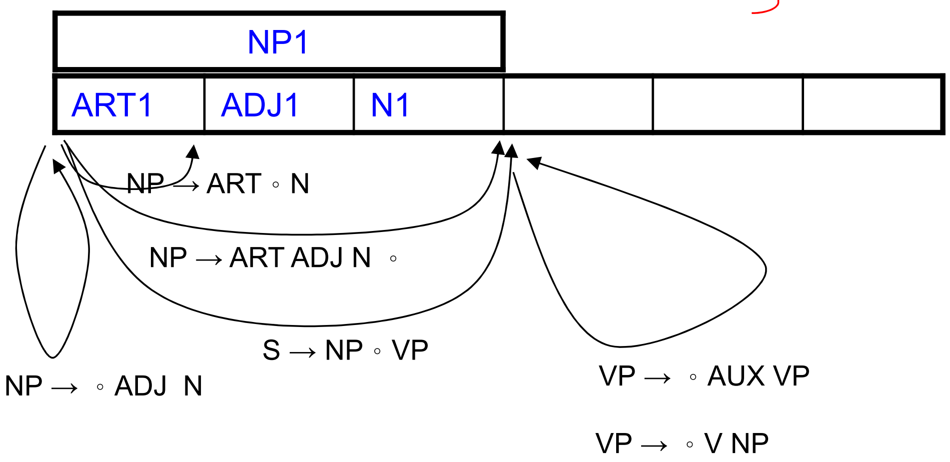
Loop 6

Enter N2 (*"holds"* from 4 to 5):

(using the arc extension algorithm)
(using the arc introduction algorithm)

No Operation

Chart:



Agenda:

~~N2 4,5~~
V1 4,5

The Top-Down Chart Parser (16/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

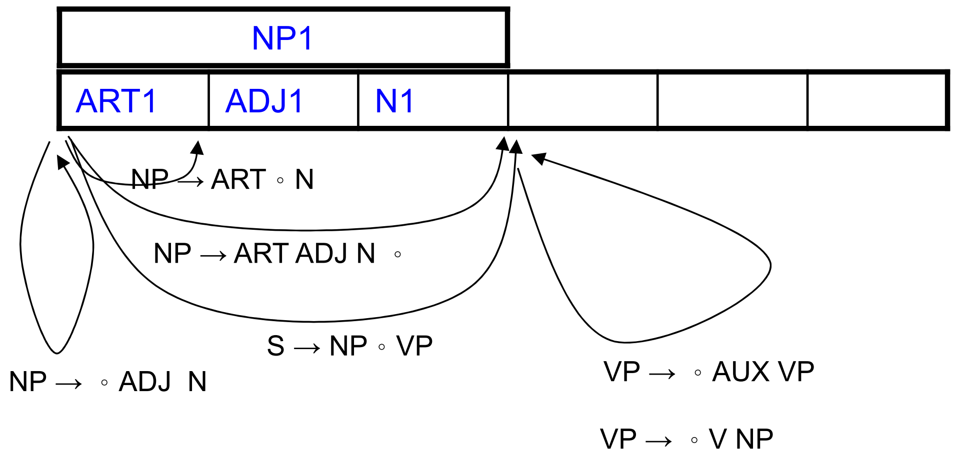
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 7

Enter V1 (*"holds"* from 4 to 5):

Chart:



Agenda:

V1 4,5

The Top-Down Chart Parser (17/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

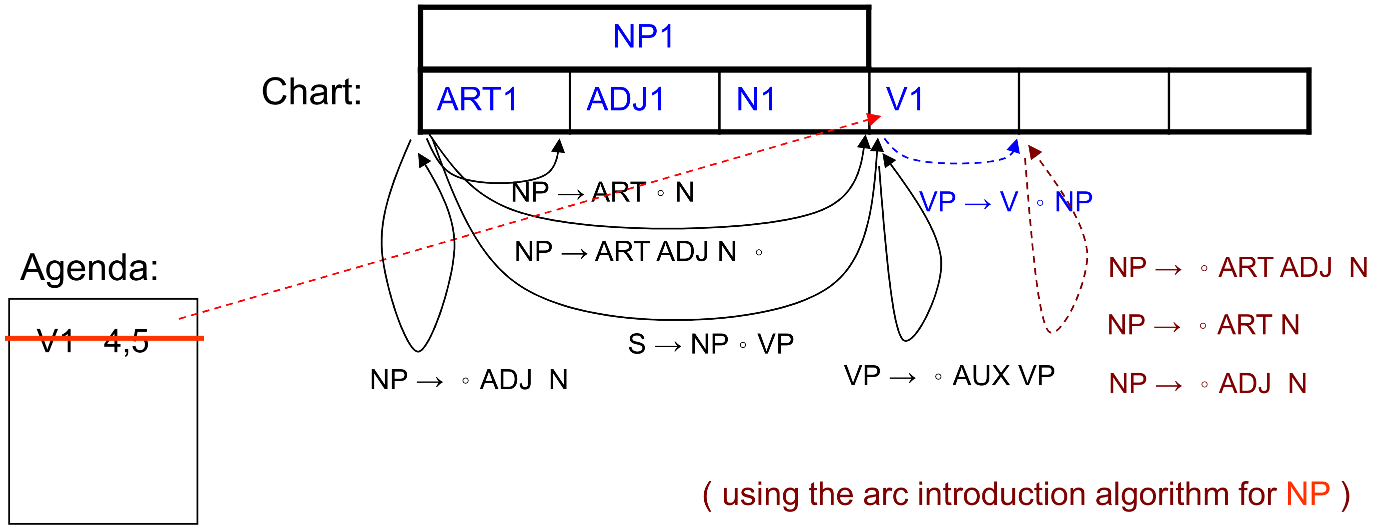
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 7

Enter V1 (*"holds"* from 4 to 5):

(using the arc extension algorithm)



(using the arc introduction algorithm for NP)

The Top-Down Chart Parser (18/27)

- Example $\quad {}_1$ The ${}_2$ large ${}_3$ can ${}_4$ holds ${}_5$ the ${}_6$ water ${}_7$

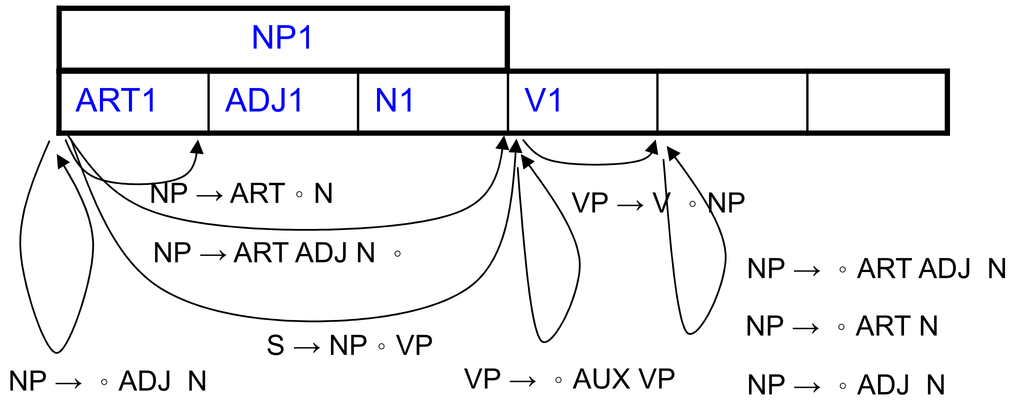
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 8

Enter ART2 (*"the"* from 5 to 6): Look at next word

Chart:



Agenda:

ART2 5,6

The Top-Down Chart Parser (19/27)

- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

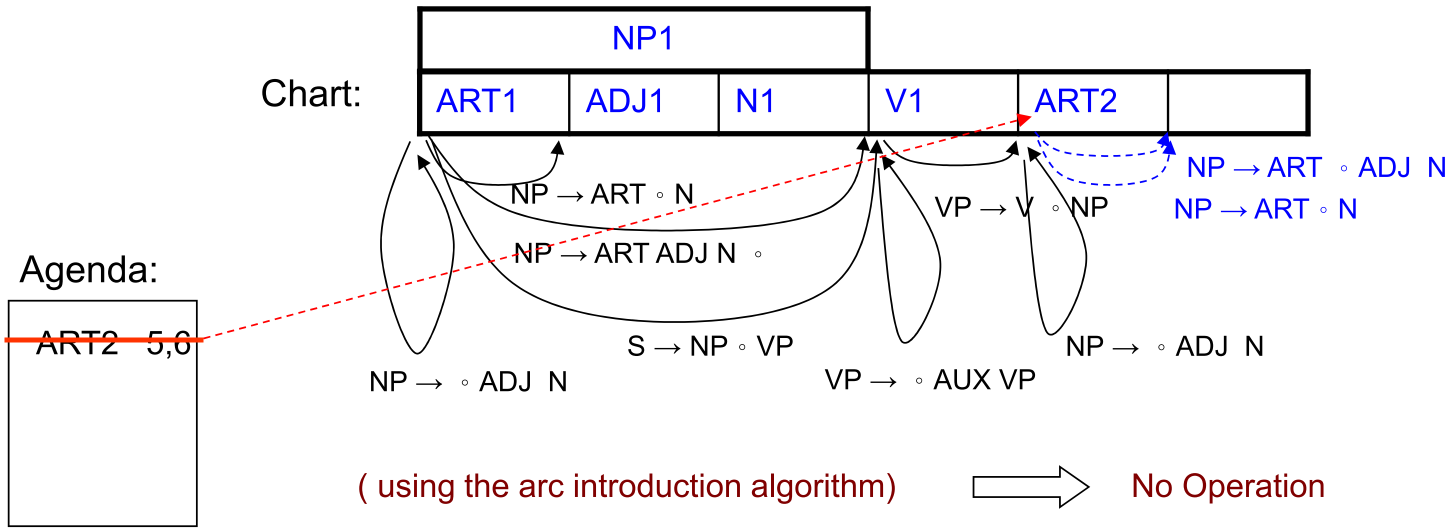
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 8

Enter ART2 (*"the"* from 5 to 6):

(using the arc extension algorithm)



(using the arc introduction algorithm) ⇨ No Operation

The Top-Down Chart Parser (20/27)

- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

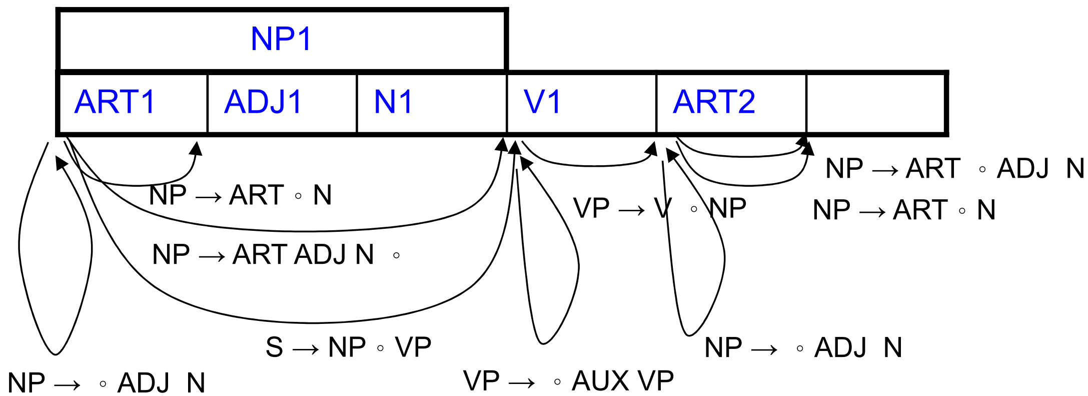
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 9

Enter N3 (*"water"* from 6 to 7): Look at next word

Chart:



Agenda:

N3 6,7

The Top-Down Chart Parser (21/27)

- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

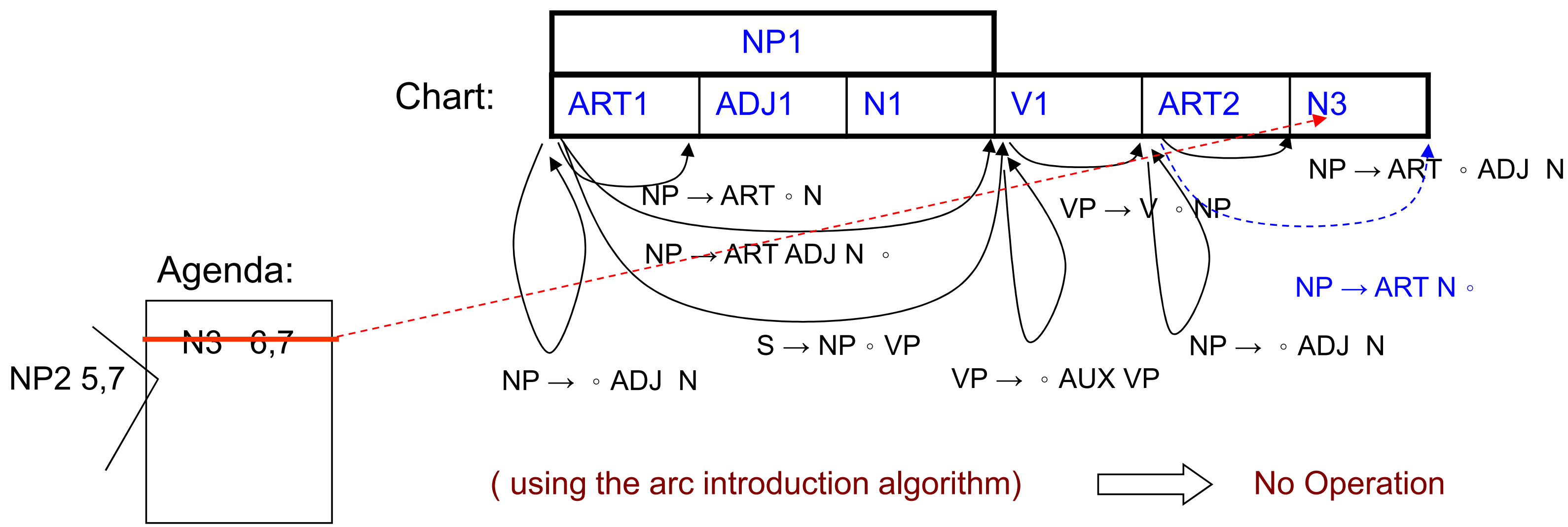
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 9

Enter N3 (“*water*” from 6 to 7):

(using the arc extension algorithm)



The Top-Down Chart Parser (22/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

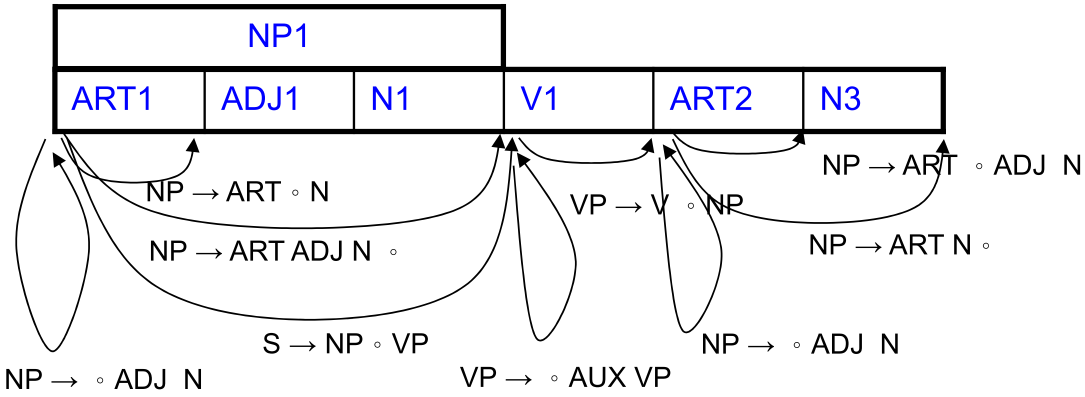
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 10

Enter NP2 (*"the water"* from 5 to 7):

Chart:



Agenda:

NP2	5,7

The Top-Down Chart Parser (23/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

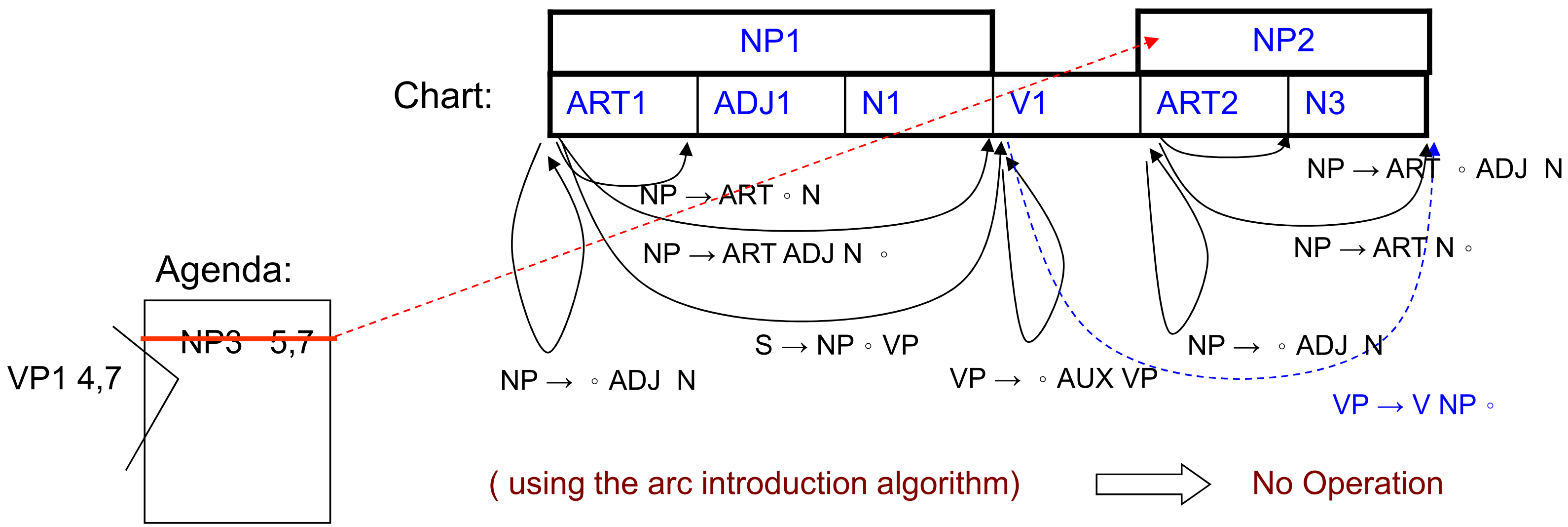
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 10

Enter NP2 (*"the water"* from 5 to 7):

(using the arc extension algorithm)



(using the arc introduction algorithm) ⇨ No Operation

The Top-Down Chart Parser (24/27)

- Example $\quad {}_1$ The ${}_2$ large ${}_3$ can ${}_4$ holds ${}_5$ the ${}_6$ water ${}_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

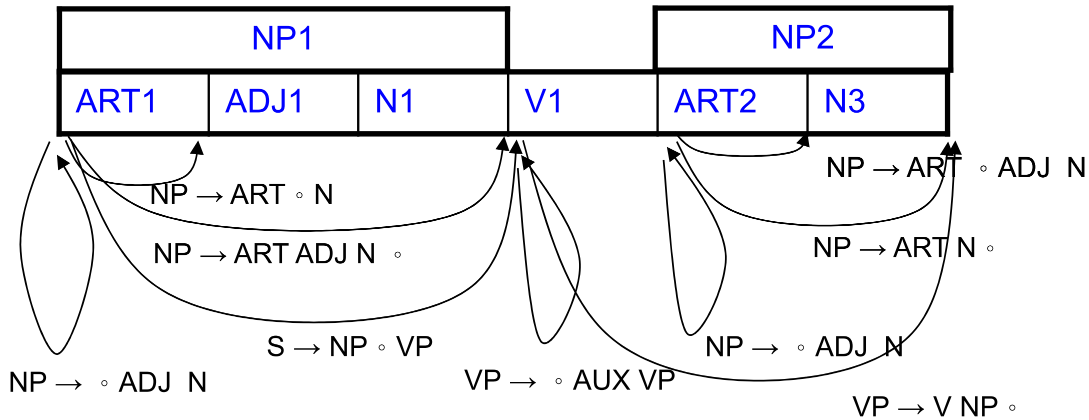
Loop 11

Enter VP1 (*“holds the water”* from 4 to 7):

Agenda:

VP1 4,7

Chart:



The Top-Down Chart Parser (25/27)

- Example

$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

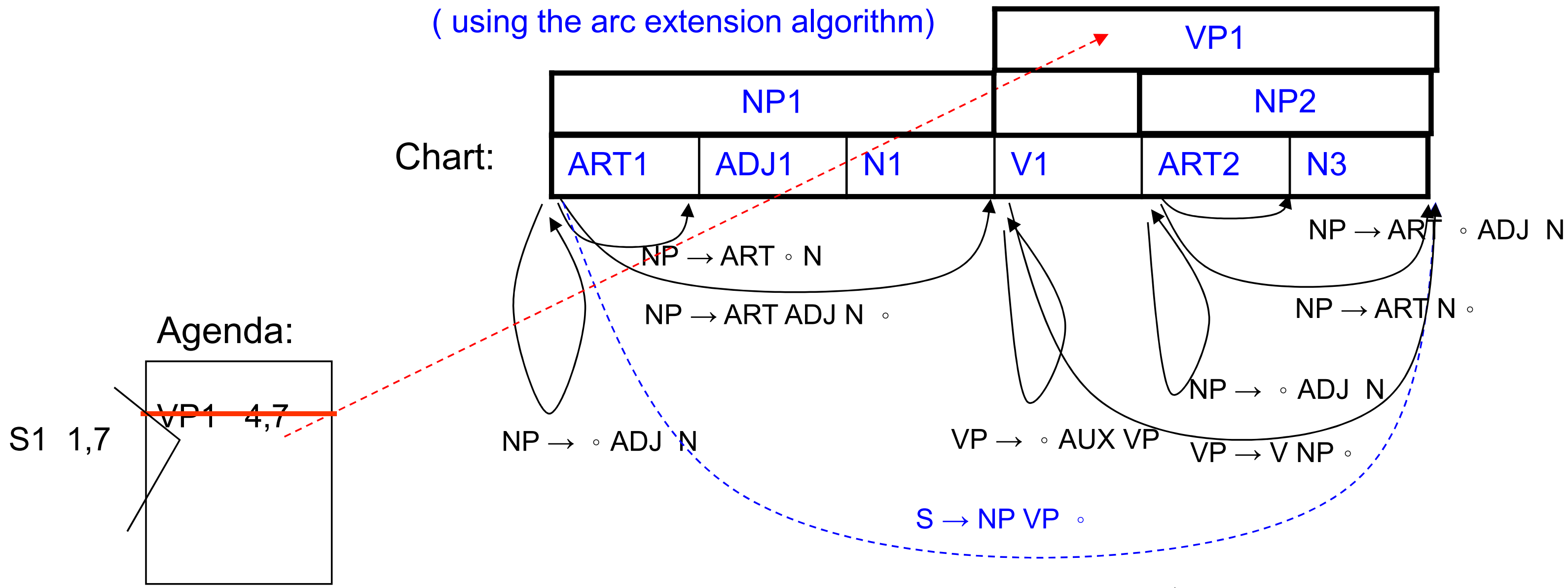
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 11

Enter VP1 (*"holds the water"* from 4 to 7):

(using the arc extension algorithm)



(using the arc introduction algorithm) ⇒ No Operation

The Top-Down Chart Parser (26/27)

- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

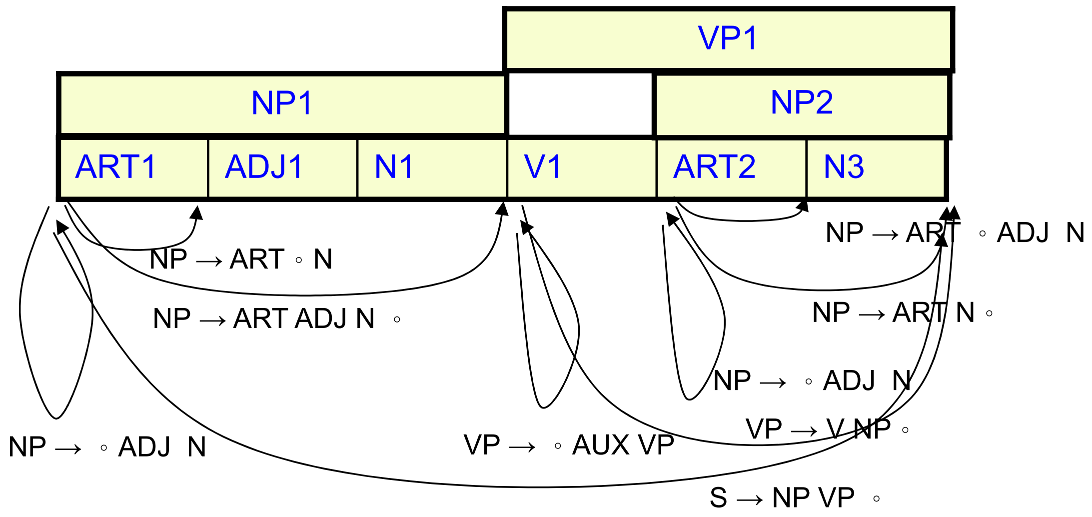
Loop 12

Enter S1 (*"the large can holds the water"* from 1 to 7):

Agenda:

S1 1,7

Chart:



The Top-Down Chart Parser (27/27)

- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

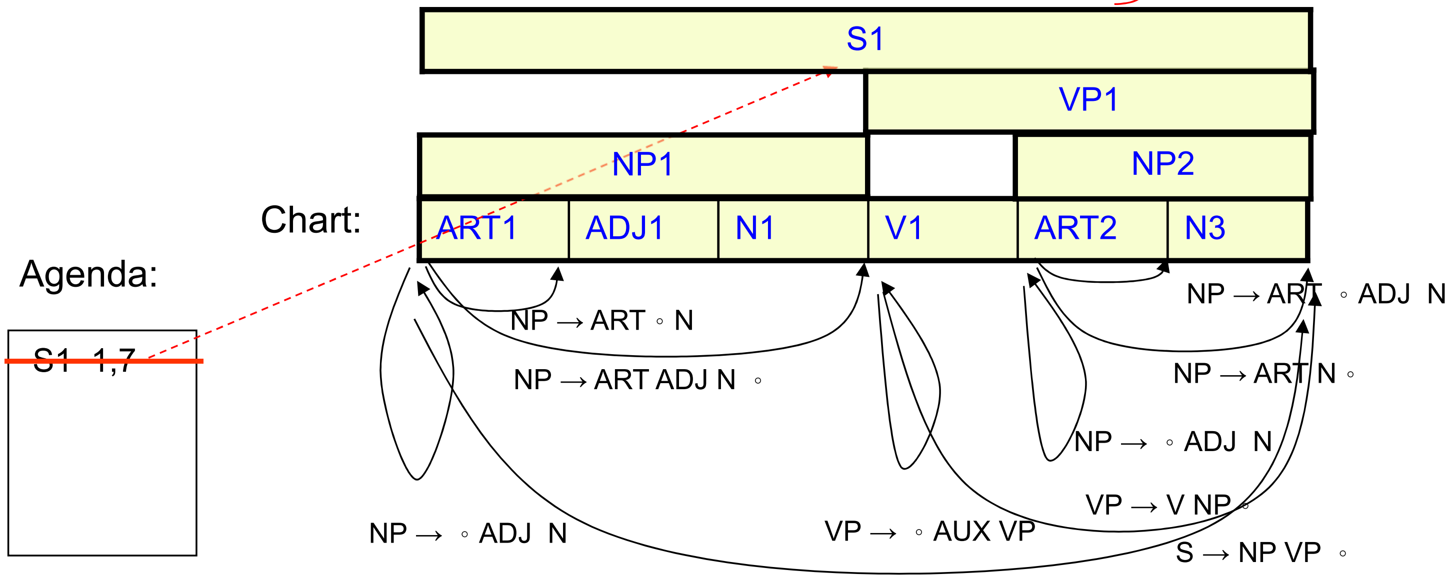
Loop 12

Enter S1 (*“the large can holds the water”* from 1 to 7):

(using the arc extension algorithm)

(using the arc introduction algorithm)

No Operation



Comparison between the Bottom-Up an Top-Down Chart Parsers

- The number of constituents generated by the top-down chart parser has dropped from 15 to 10
- In practice, the top-down method is considerably more efficient for any reasonable grammar