Parsing with Context-Free Grammars

Berlin Chen 2003

References:

1. Speech and Language Processing, chapters 9, 10
2. Natural Language Understanding, chapter 3
3. Jim Martin's Lecture Notes

Context-Free Grammars (CFGs)

- Formalized by Chomsky(1956), and Backus (1959)
  - Also called Backus-Naur Form (BNF)
- Also called phrase-structure grammars
- 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 we say about how the various kinds of units behave

Major Characteristics of CFGs

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

$NP \rightarrow Det\ Nominal$

$NP \rightarrow ProperNoum$

$Nominal \rightarrow Noun \mid Noun\ Nominal$

$Det \rightarrow a$

$Det \rightarrow the$

$Noun \rightarrow flight$

mother

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

Major Characteristics of CFGs

- 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
    - The non-terminal symbol can be a larger constituent (e.g. a phrasal unit) in addition to the lexical category

Major Characteristics of CFGs

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

- CFGs can be thought of 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 sentences given a grammar
    - Top-down or bottom-up strategies

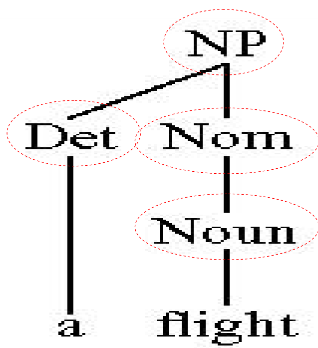
Major Characteristics of CFGs

- 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 productions P, each of the form $A \rightarrow \alpha$, where A is a non-terminal symbol and α 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
  – Via the concept of "*derivation*"

Derivations

- Derivation: a sequence of rules applied to a string that accounts for that string
  - Can be represented by a parse tree
  - E.g. a parse tree for "a flight"



A derivation represented by a parse tree

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

Derivations as Trees

- Directly Derive
  - Directly derive:

    $A \rightarrow \beta,\ \ \alpha A \gamma \Rightarrow \alpha \beta \gamma$

    directly derives

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

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

    derives
- **Parsing**: the process of taking a string and a grammar and returning a (or many) parse tree(s) for that string
  - Analogous to running a finite-state transducer with a tape
  - But, is it more powerful (?)

More Complex Derivations

- $S \rightarrow NP\ VP$
  - 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.

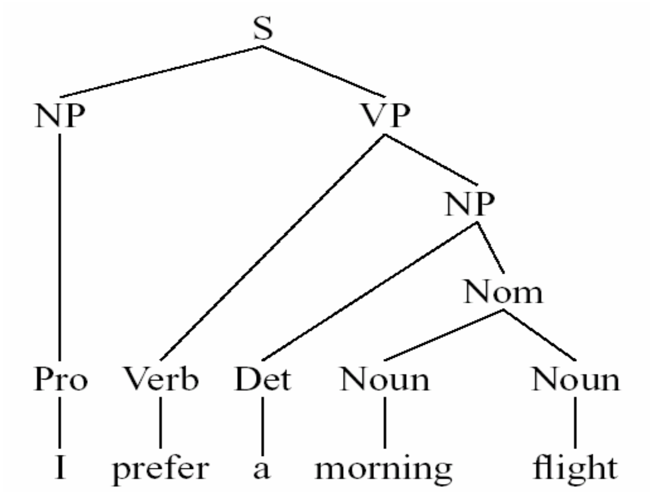
$NP \rightarrow Det\ Nominal$

$NP \rightarrow ProperNoum$

$VP \rightarrow Verb\ NP$

$VP \rightarrow Verb\ NP\ PP$

More Complex Derivations



Bracketed notation

[$_S$ [$_{NP}$[$_{Pro}$ I]][$_{VP}$[$_V$ prefer][$_{NP}$[$_{Det}$ a][$_{Nom}$ [$_N$ morning][$_N$ flight]]]]]

S	→	*NP VP*	I + want a morning flight
NP	→	*Pronoun*	I
	|	*Proper-Noun*	Los Angeles
	|	*Det Nominal*	a + flight
Nominal	→	*Noun Nominal*	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

Noun	→	*flights* | *breeze* | *trip* | *morning* | ...
Verb	→	*is* | *prefer* | *like* | *need* | *want* | *fly*
Adjective	→	*cheapest* | *non−stop* | *first* | *latest*
		| *other* | *direct* | ...
Pronoun	→	*me* | *I* | *you* | *it* | ...
Proper-Noun	→	*Alaska* | *Baltimore* | *Los Angeles*
		| *Chicago* | *United* | *American* | ...
Determiner	→	*the* | *a* | *an* | *this* | *these* | *that* | ...
Preposition	→	*from* | *to* | *on* | *near* | ...
Conjunction	→	*and* | *or* | *but* | ...

More Complex Derivations

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

    NP $\rightarrow$ NP PP [[The flight] [to Boston]]

    VP $\rightarrow$ VP PP [[departed Miami] [at noon]]
  - E.g.
    - flights from Denver
    - Flights from Denver to Miami
    - Flights from Denver to Miami in February
    - Flights from Denver to Miami in February on Friday

Sentence-level Construction of English

- Declaratives: *A plane left.*

  S → NP VP

- Imperatives: *Show the lowest fare.*

  S → VP

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

  S → Aux NP VP

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

  S → WH Aux NP VP

Parsing Strategies

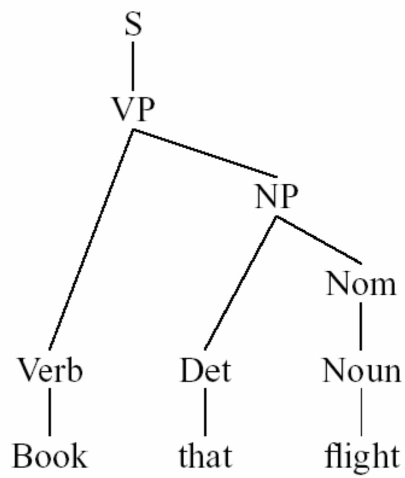
- Top-Down Parsing
  - Start with the *S* symbol and search through different ways to rewrite the symbols until the input sentence is generated, or until all possibilities have been explored

- 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

Parsing as Search

– Different search algorithms, such as depth-first search or breadth-first search 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. S → NP VP
2. NP → ART N
3. NP → ART ADJ N
4. VP → V
5. VP → V NP

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

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

The Simple Top-Down Parser

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

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

Check the input sentence
o see if match ASAP

new states are put onto the
front of the possibilities list

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

cried: V
dogs: N, V
the: ART

The Simple Top-Down Parser

- Algorithm
  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
  2. If C consists of an empty symbol list and is at the sentence end position, the algorithm succeeds
  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 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, generate a new state for each rule in the grammar that can rewrite that non-terminal symobl and **add them all to the possibilities list**

where to put the new states depends on the search strategies

The Simple Top-Down Parser

- One more example

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

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

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

new states are put onto the front of 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

- 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

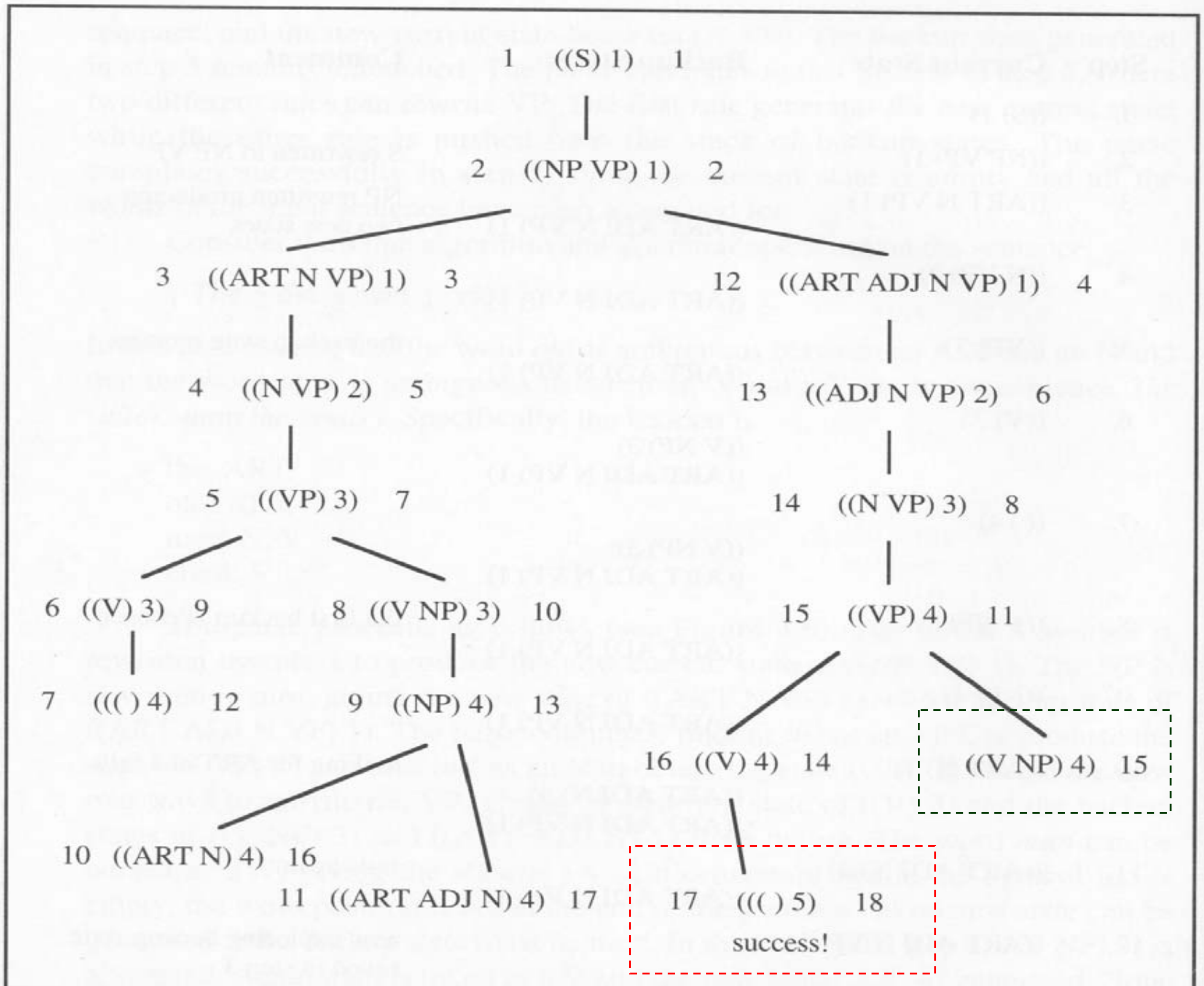


Figure 3.7 Search tree for two parse strategies (depth-first strategy on left; breadth-first on right)

Search strategies for the Top-Down Parser

- 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

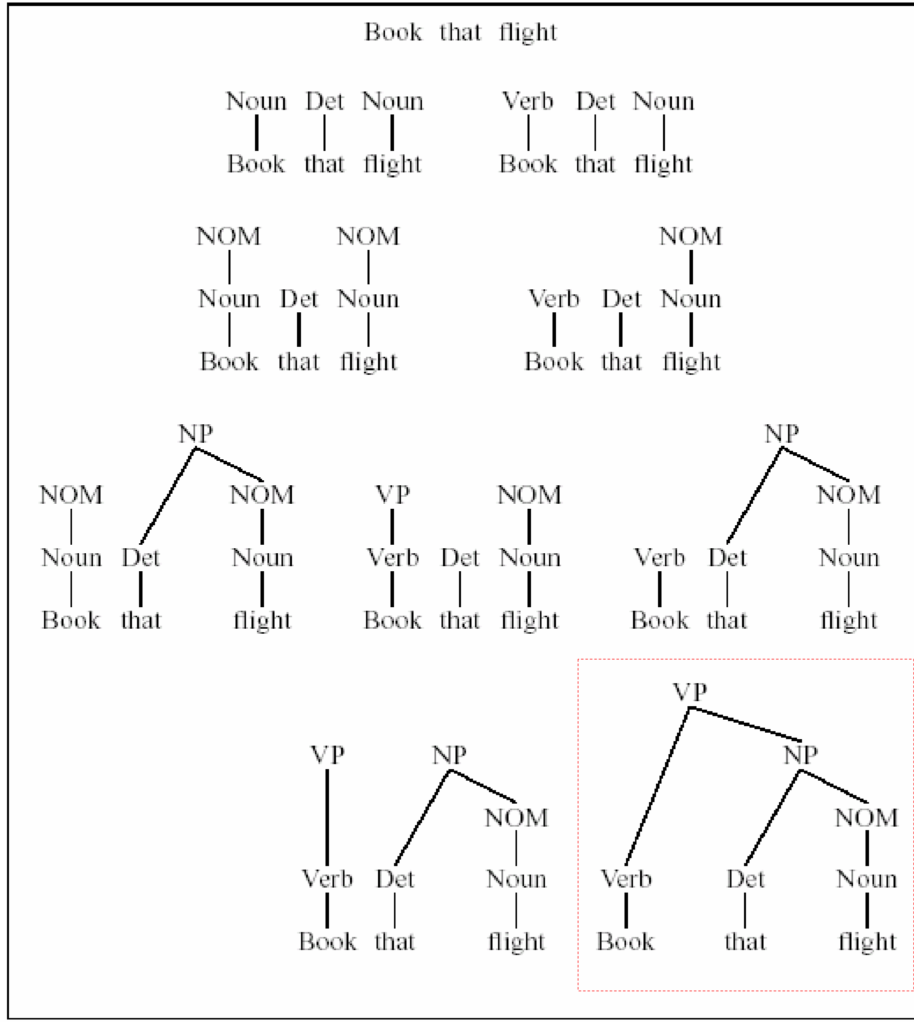
The Bottom-Up Parser

- 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

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

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



Comparing Top-Down and Bottom-UP Parsing

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

Problems with Parsing

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

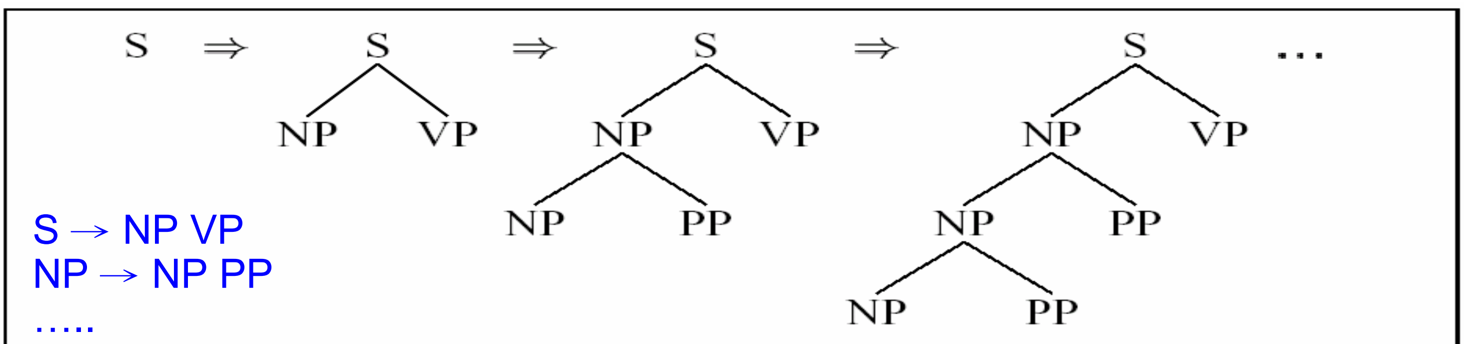
    NP → Det Nominal

    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

- **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

- **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

- Repeated Parsing of Subtrees
  - The parser often build valid trees for portions of the input, then discards them during the backtracking, only to find that it has to rebuild them again
    - Some constituents are constructed more than once

The Bottom-Up Chart Parser

- 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
  - Three kinds of data structures
    - The agenda
    - 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 with respect to the input

The Bottom-Up Chart Parser

- 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 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**

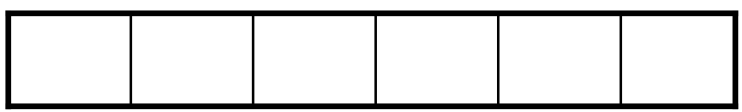
The Bottom-Up Chart Parser

- 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
hold: N, V
Water: N

Initialization

Chart: 

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

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

The Bottom-Up Chart Parser

- 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

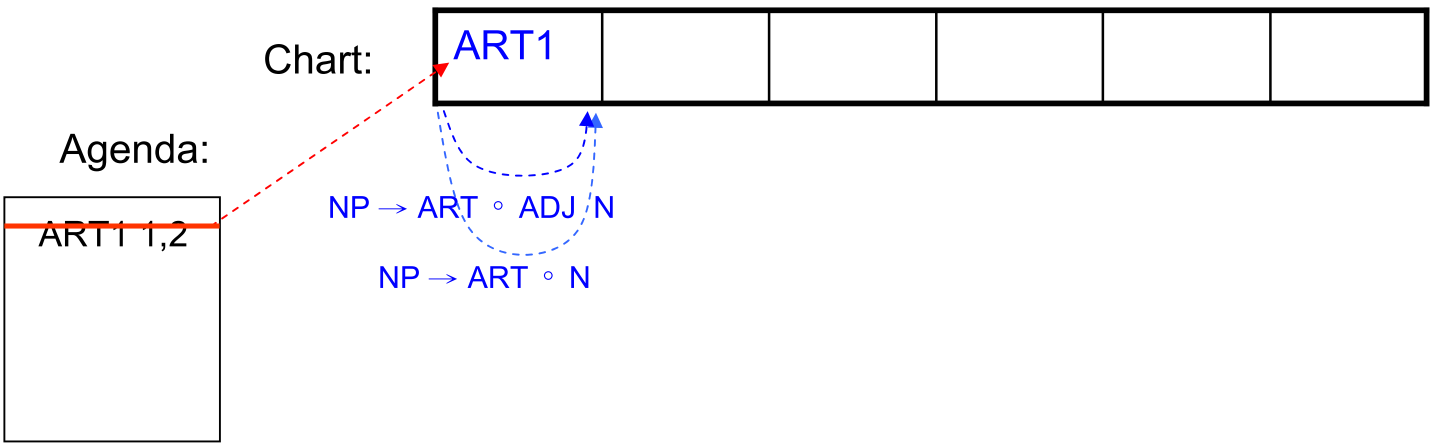
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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 Bottom-Up Chart Parser

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

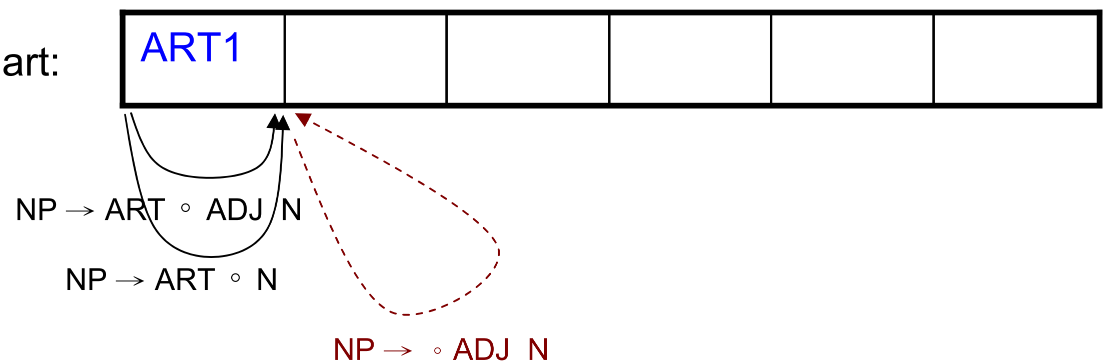
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Loop 2 Look at next word

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

Chart:



Agenda:

ADJ1 2,3

The Bottom-Up Chart Parser

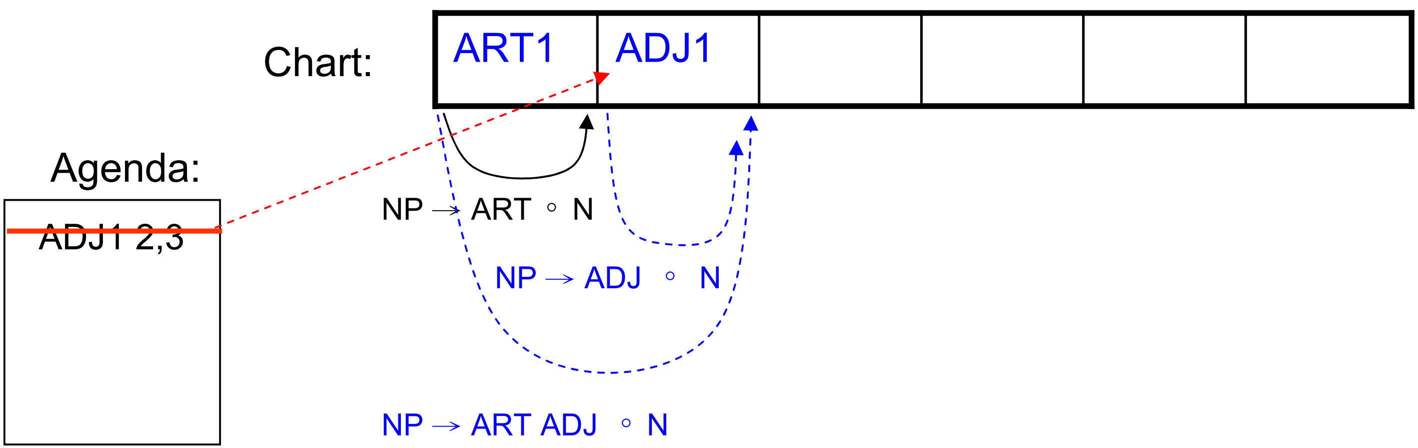
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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)



The Bottom-Up Chart Parser

- 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

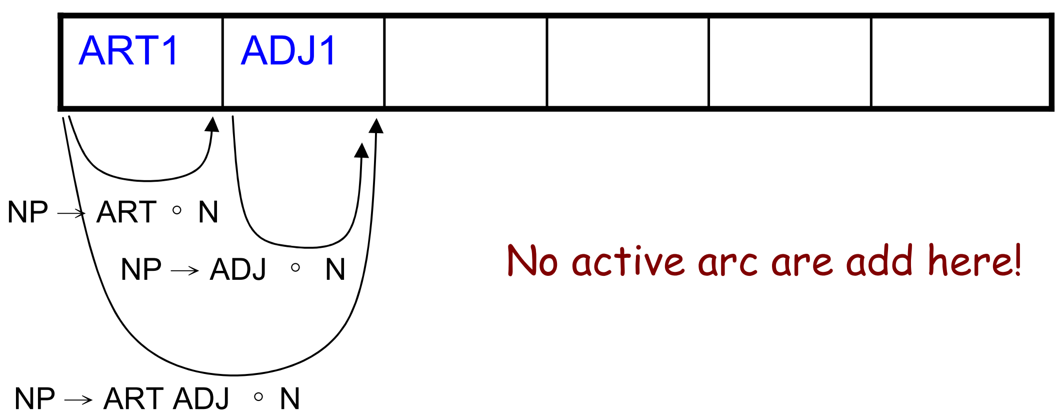
the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Look at next word

Loop 3

Enter N1: (“*can*” from 3 to 4)

Chart:



No active arc are add here!

Agenda:

N1 3,4
AUX1 3,4

The Bottom-Up Chart Parser

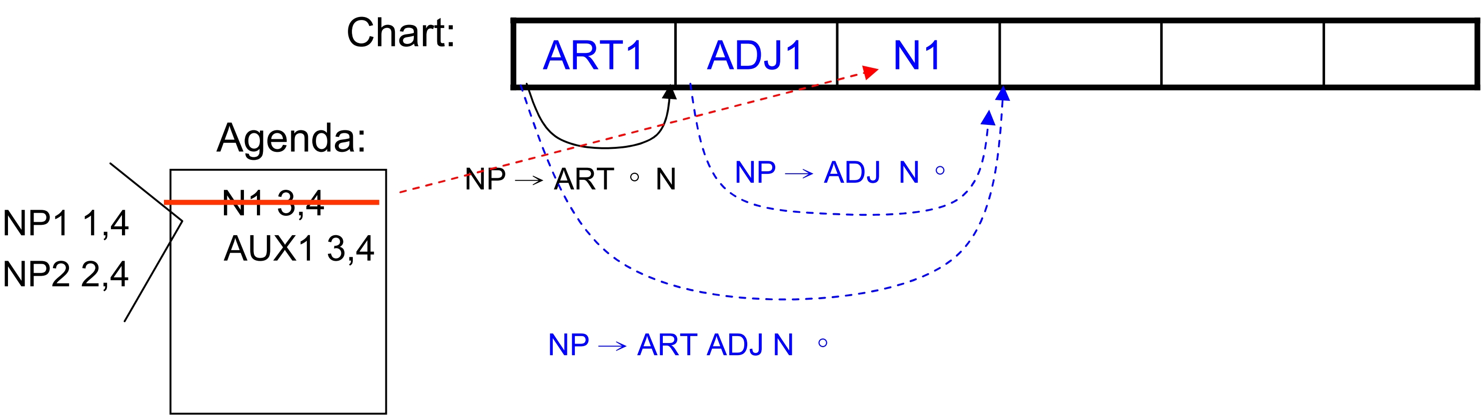
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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 (using the arc extension algorithm)

Enter N1: (“*can*” from 3 to 4)



The Bottom-Up Chart Parser

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

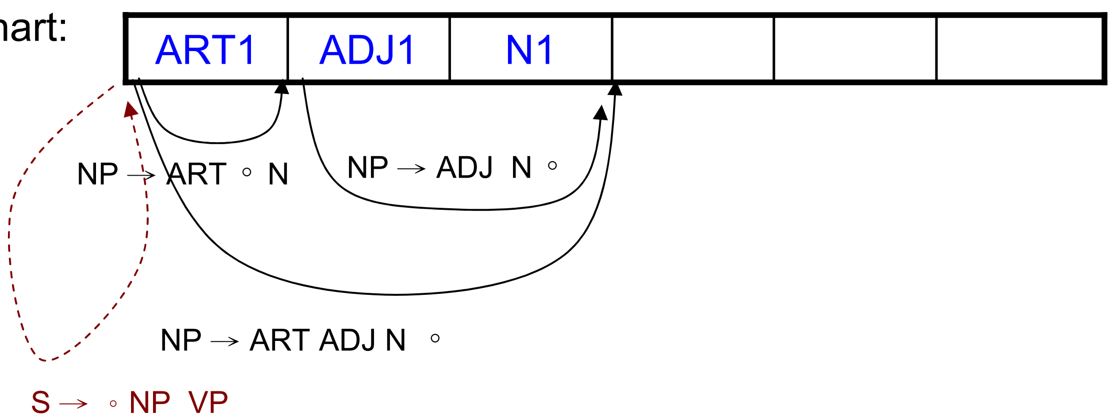
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

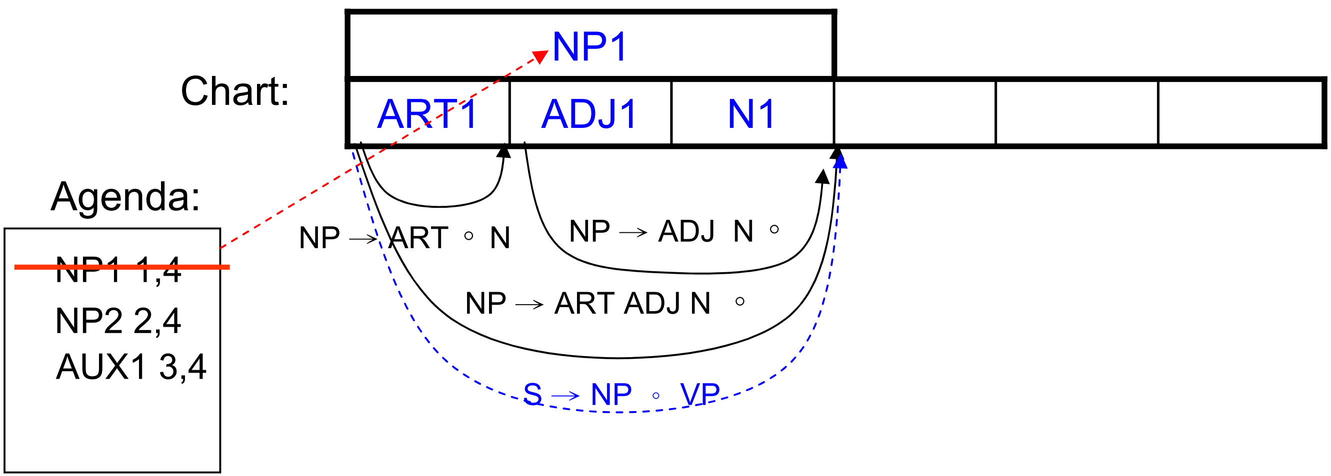
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

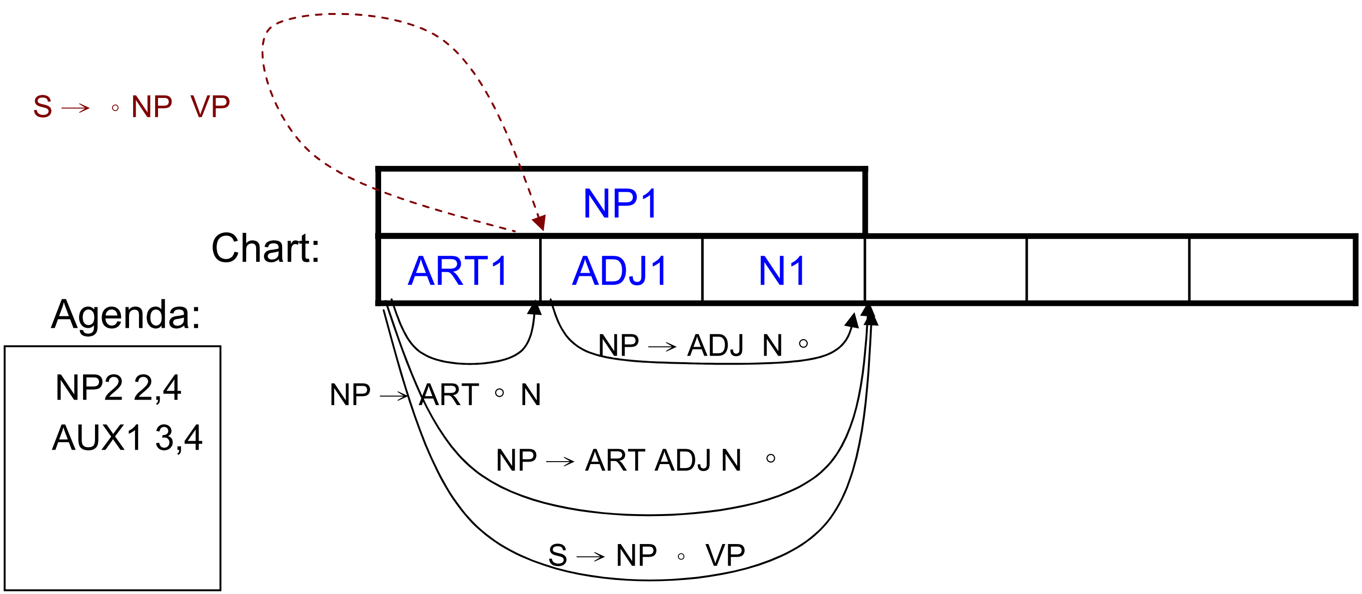
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

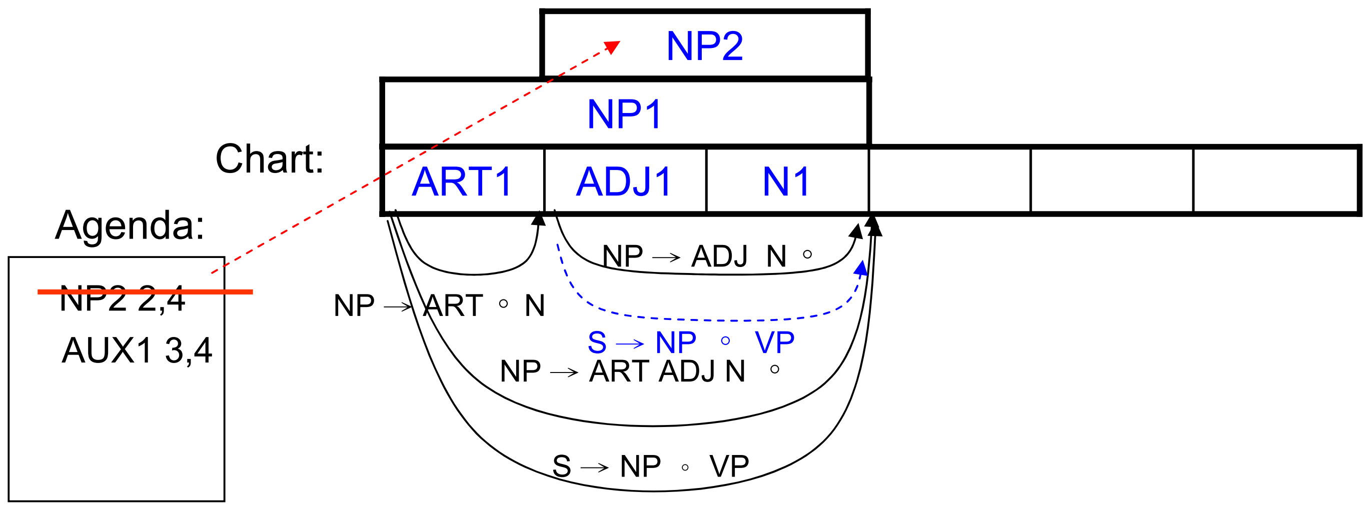
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

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

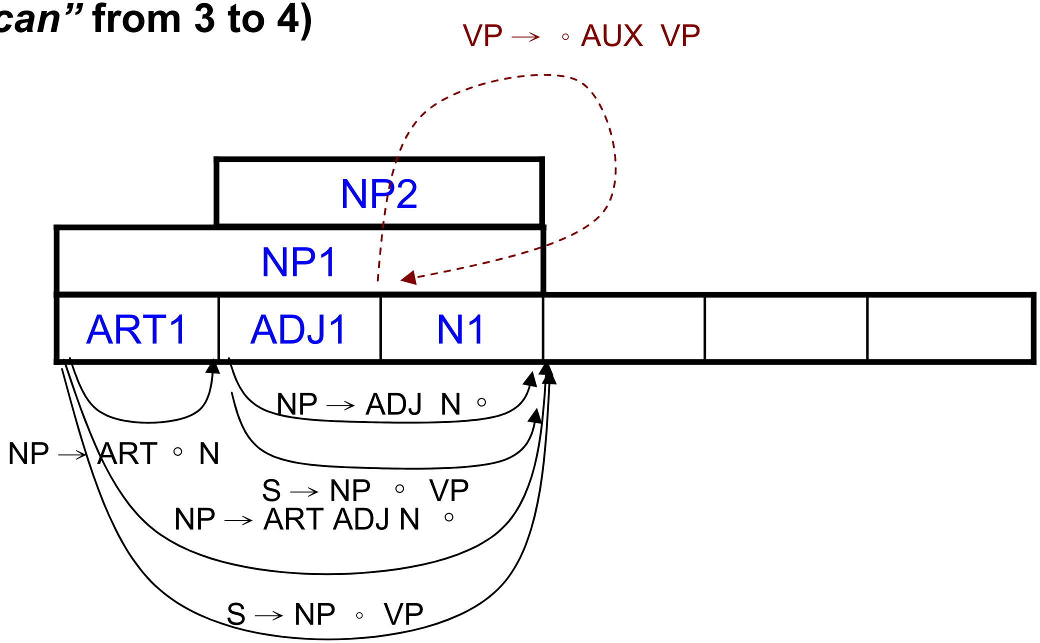
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Loop 6

Enter AUX1: (“*can*” from 3 to 4)

Chart:



Agenda:

AUX1 3,4

The Bottom-Up Chart Parser

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

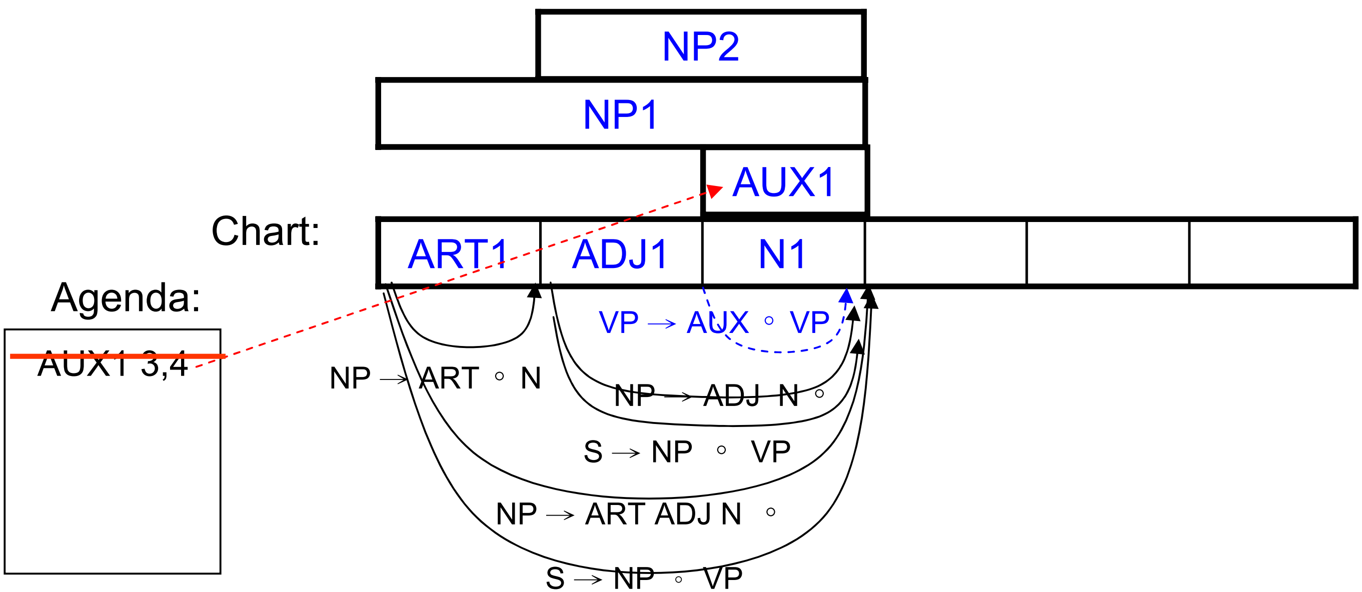
- Example

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
hold: N, V
Water: N

Loop 6 (using the arc extension algorithm)

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



The Bottom-Up Chart Parser

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

- Example

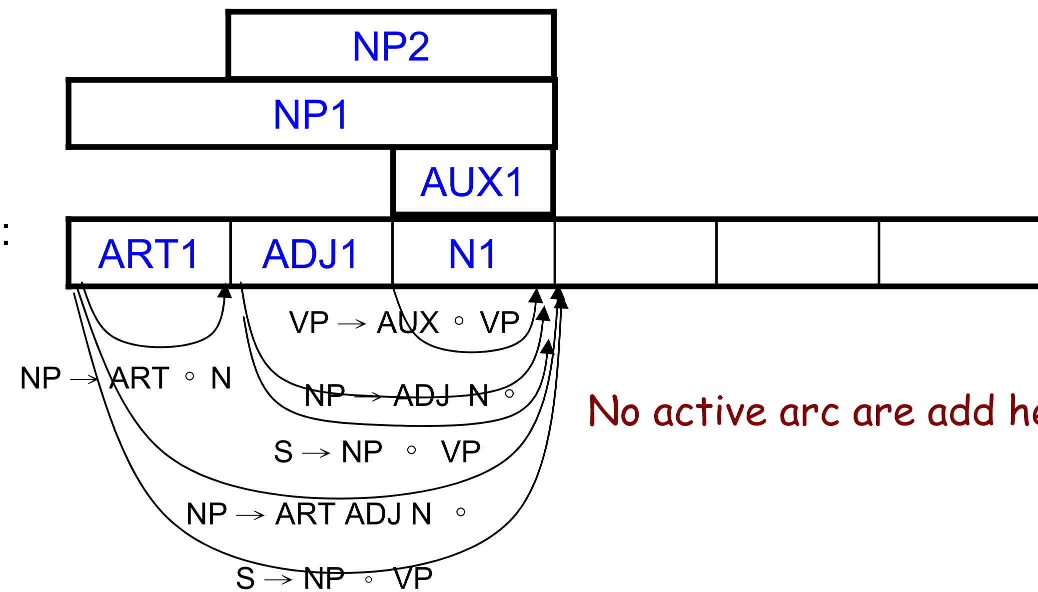
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Loop 7 Look at next word

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

Chart:



No active arc are add here!

Agenda:

N2 4,5
V1 4,5

The Bottom-Up Chart Parser

- Example

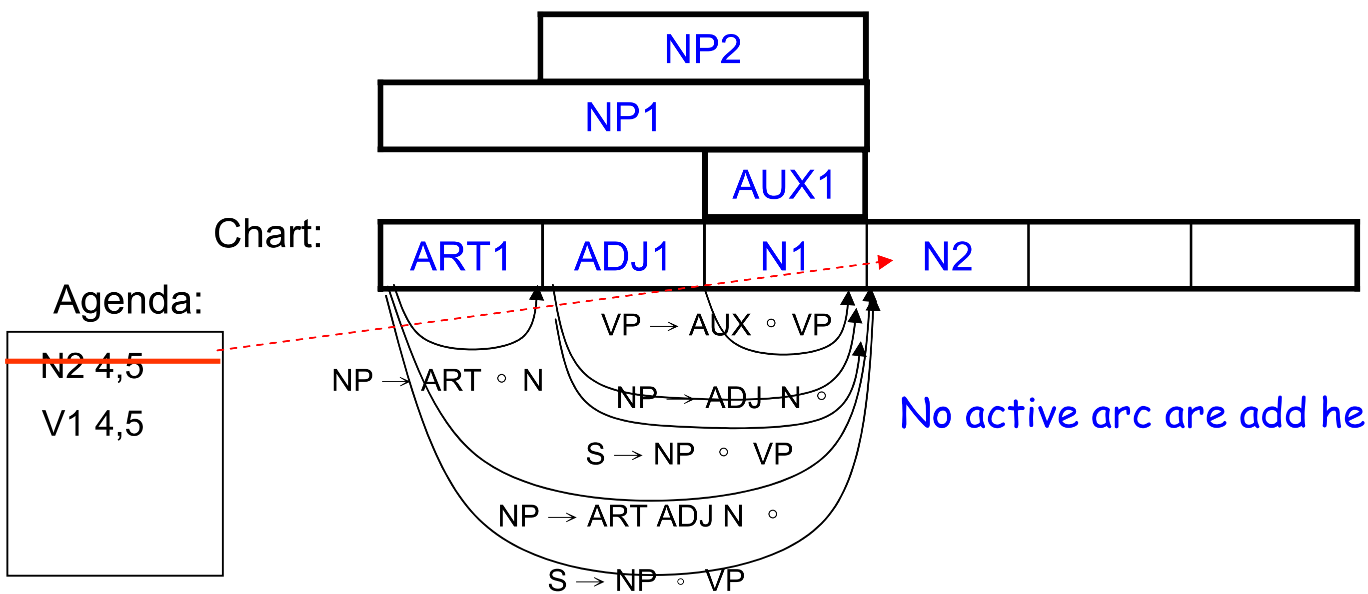
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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 arc are add here!

The Bottom-Up Chart Parser

- Example

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

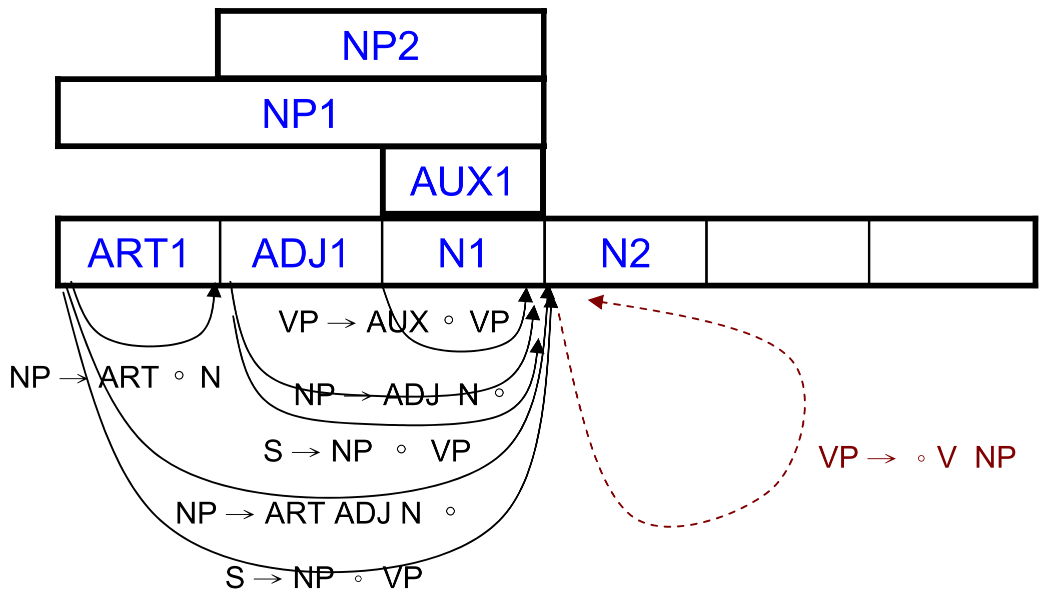
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Loop 8

Enter V1: (*“hold”* from 4 to 5)

Chart:



Agenda:

V1 4,5

The Bottom-Up Chart Parser

- Example

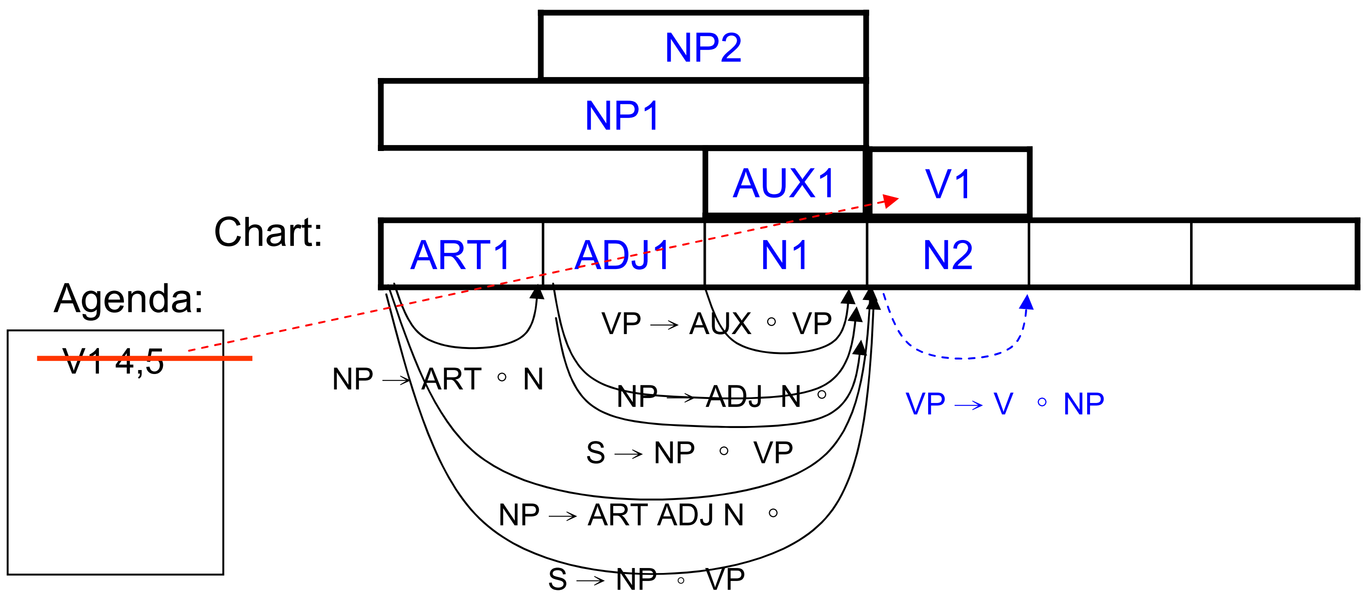
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

- Example

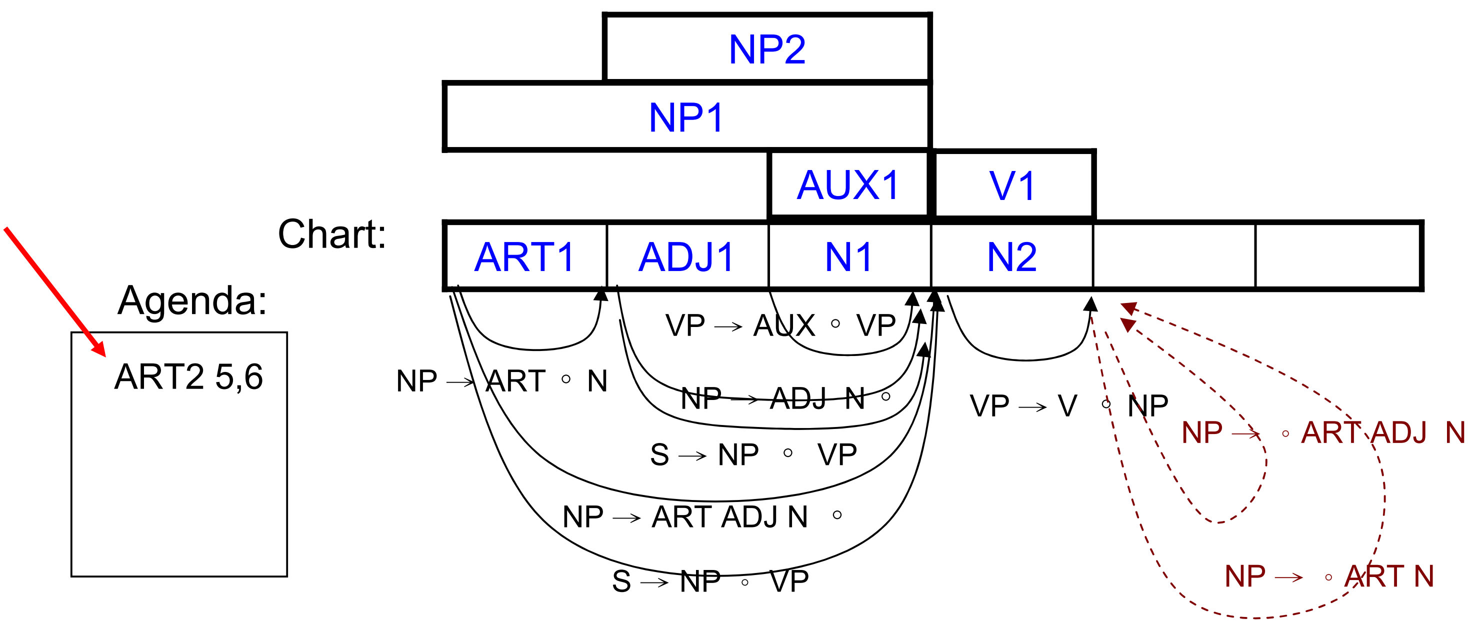
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

- Example

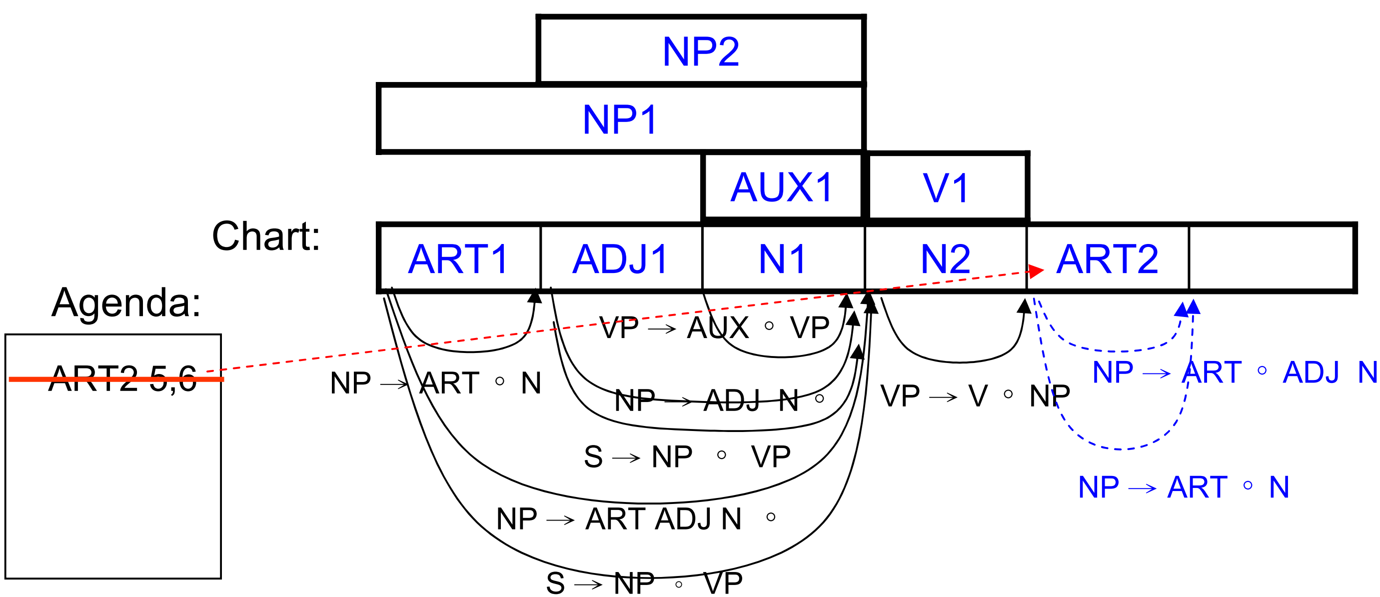
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

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

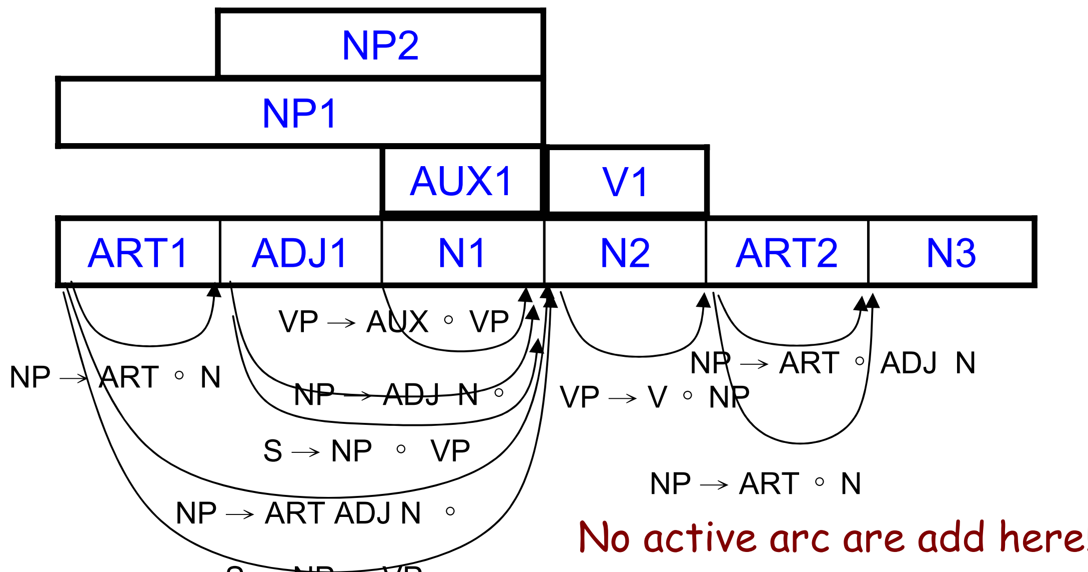
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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)

Chart:



Agenda:

N3 6,7

The Bottom-Up Chart Parser

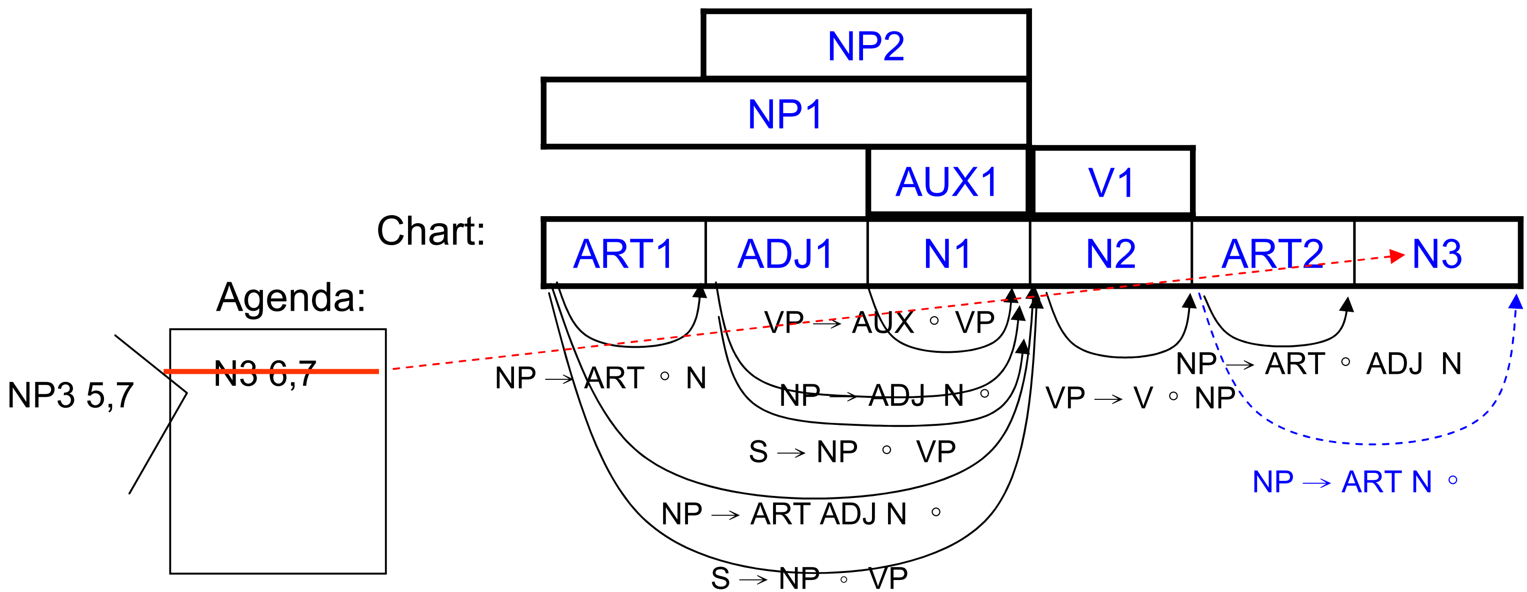
- Example $\;_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

- Example

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

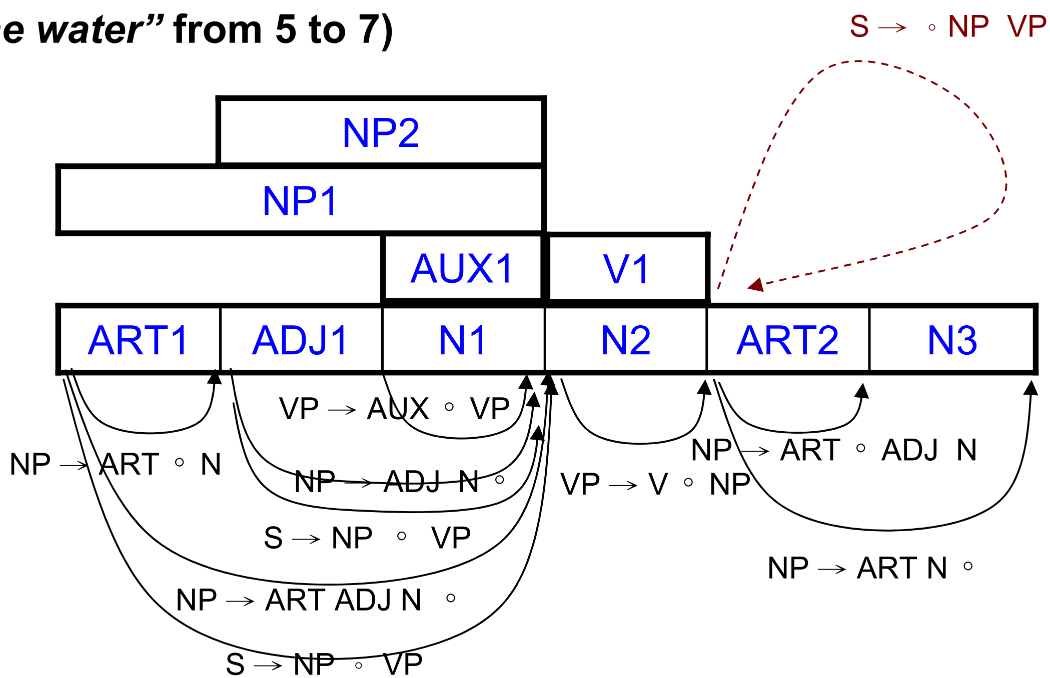
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

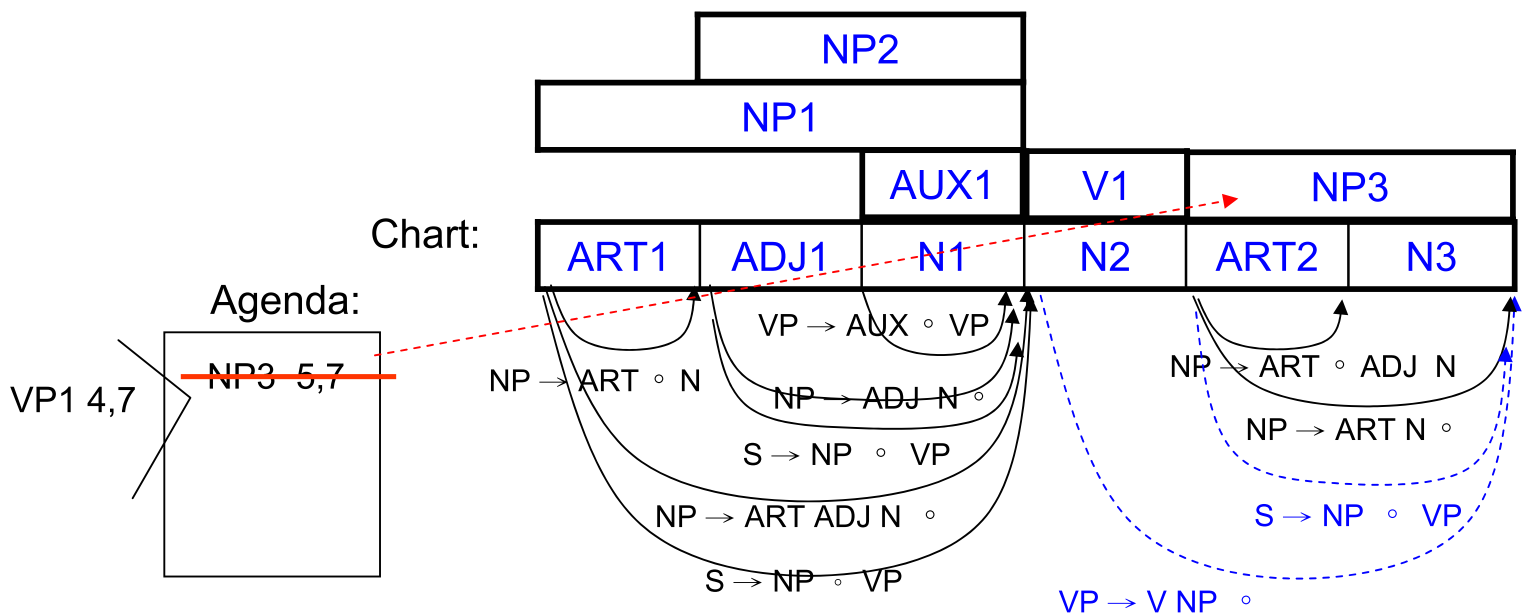
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

- 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

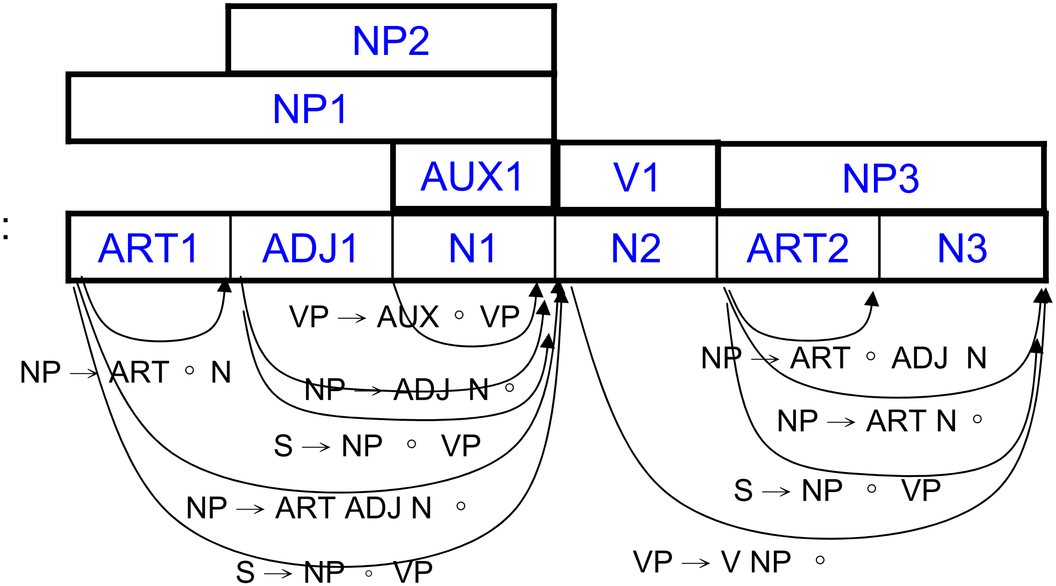
Loop 12

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

Agenda:

VP1 4,7

Chart:



No active arc are add here!

The Bottom-Up Chart Parser

- Example

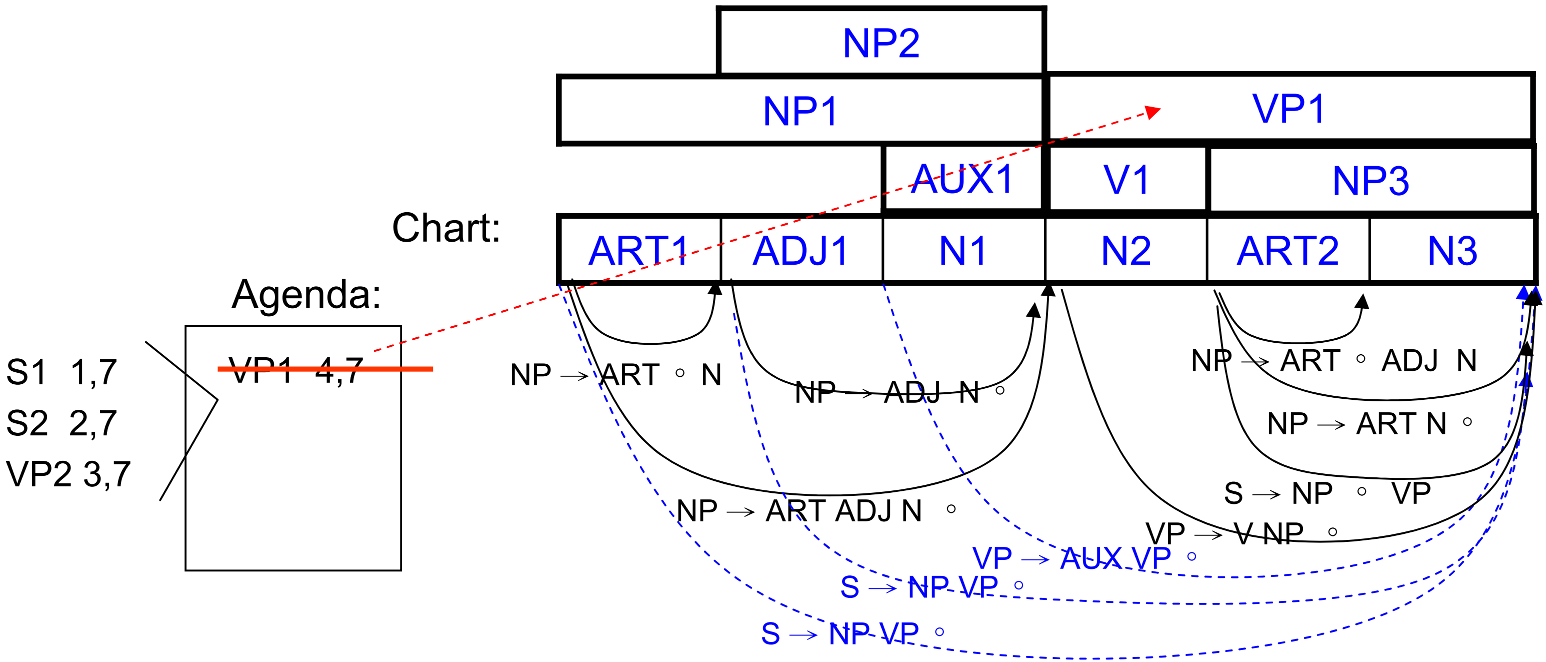
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

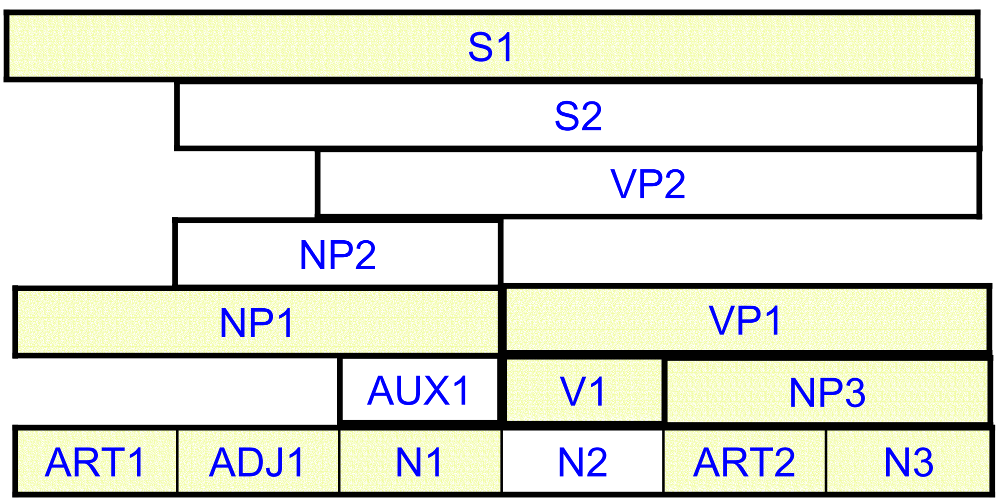
- Example

$_1$ 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



The Bottom-Up Chart Parser

- 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

The Top-Down Chart Parser

- 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 **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 $C_1 \rightarrow X_1 ... X_k$, recursively add the new arc $C_1 \rightarrow \circ X_1 ... X_k$ from position j to j

The Top-Down Chart Parser

- 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

- 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

- 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

- Example

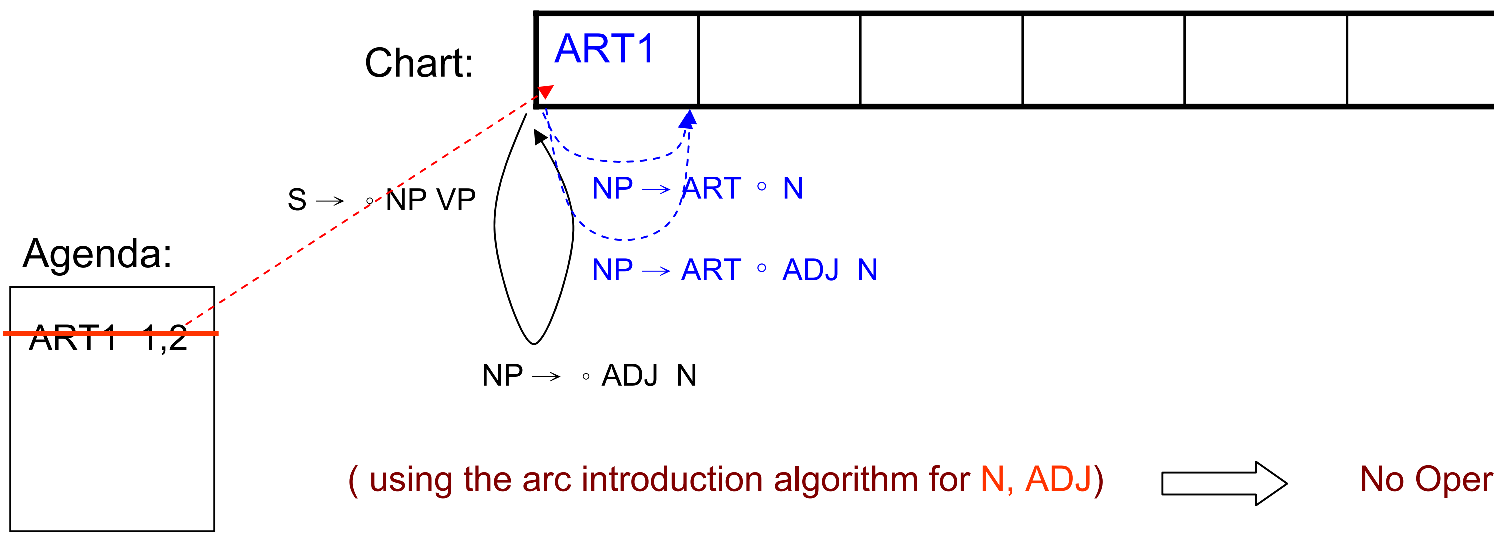
$_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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)



(using the arc introduction algorithm for N, ADJ) ⇨ No Operation

The Top-Down Chart Parser

- Example

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

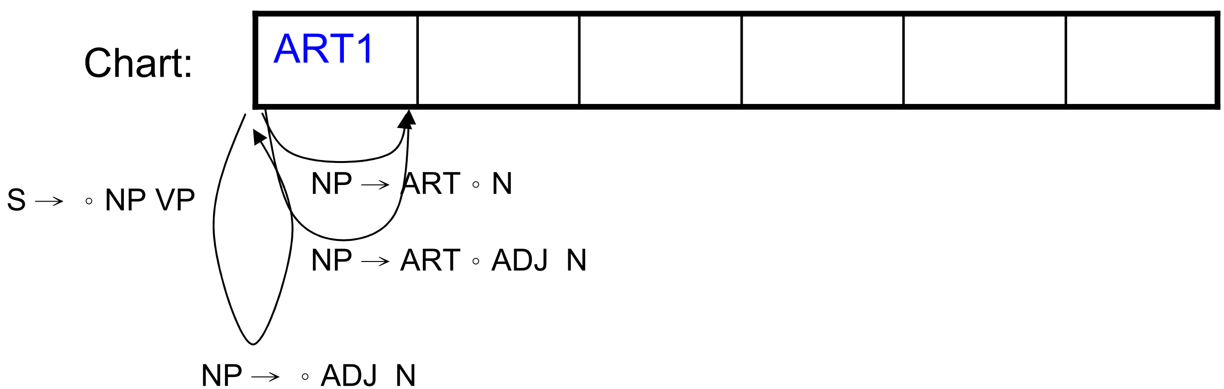
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Chart:



Agenda:

ADJ1 2,3

The Top-Down Chart Parser

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

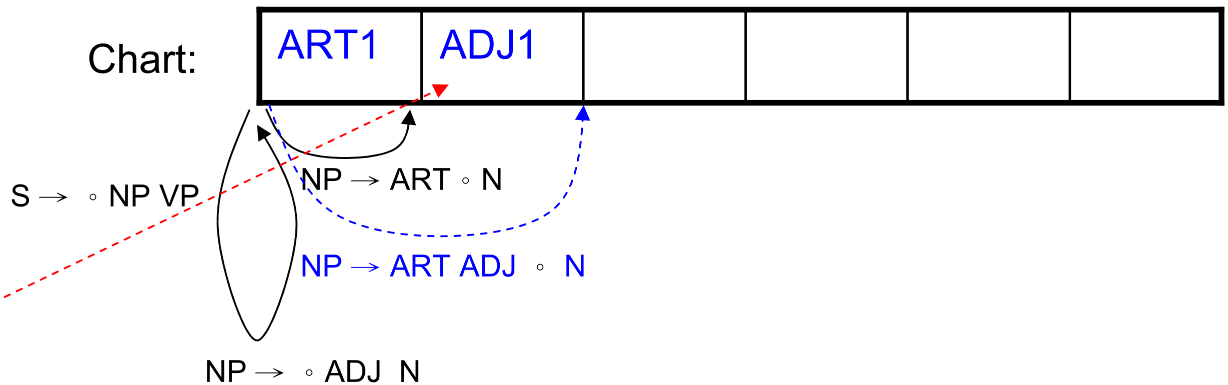
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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)

Chart:



Agenda:

~~ADJ1 2,3~~

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

The Top-Down Chart Parser

- Example

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

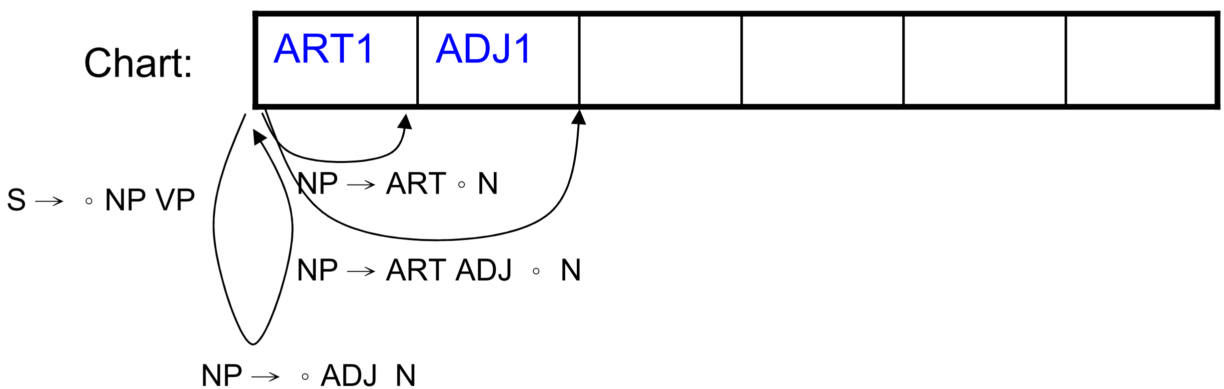
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Chart:



Agenda:

N1 3,4
AUX1 3,4

The Top-Down Chart Parser

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

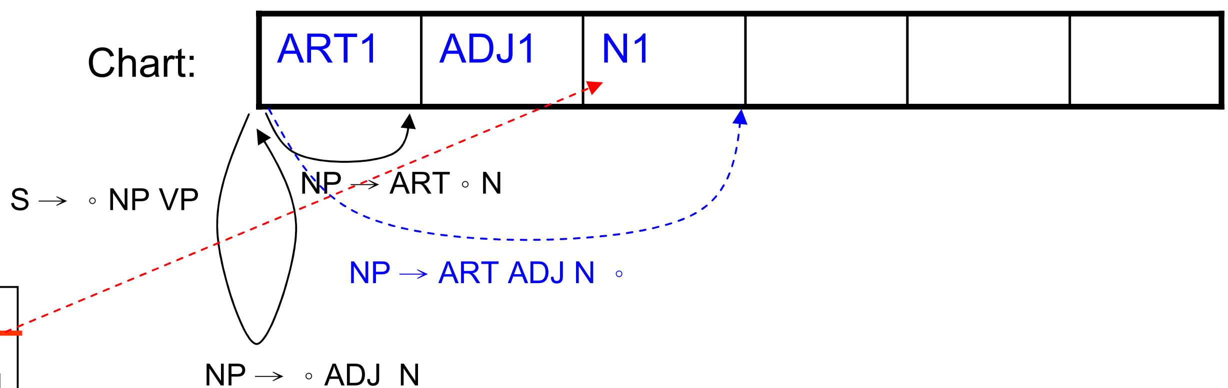
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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)

Chart:



Agenda:

NP1 1,4
~~N1 3,4~~
AUX1 3,4

(using the arc introduction algorithm) ⇨ No Operation

The Top-Down Chart Parser

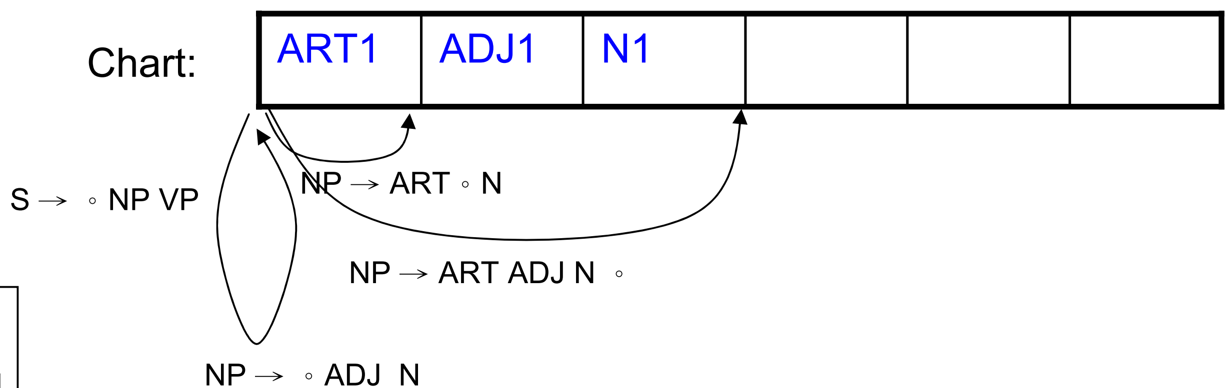
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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

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

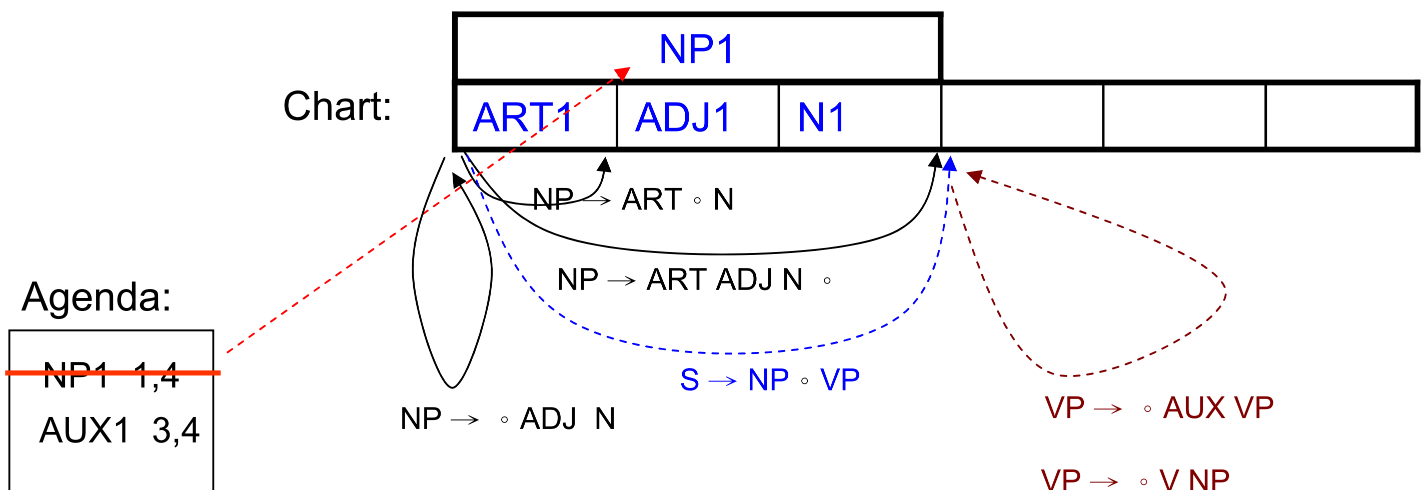
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Loop 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

- Example

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

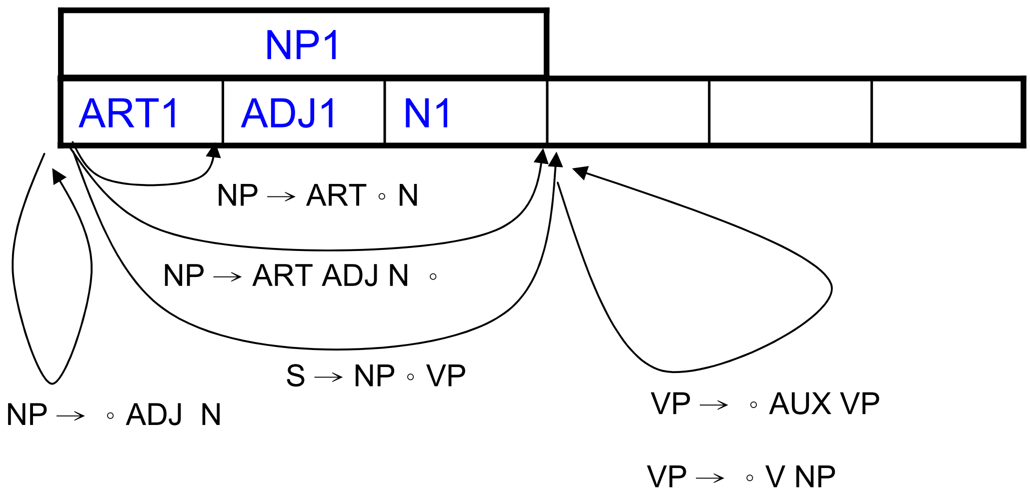
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

- 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

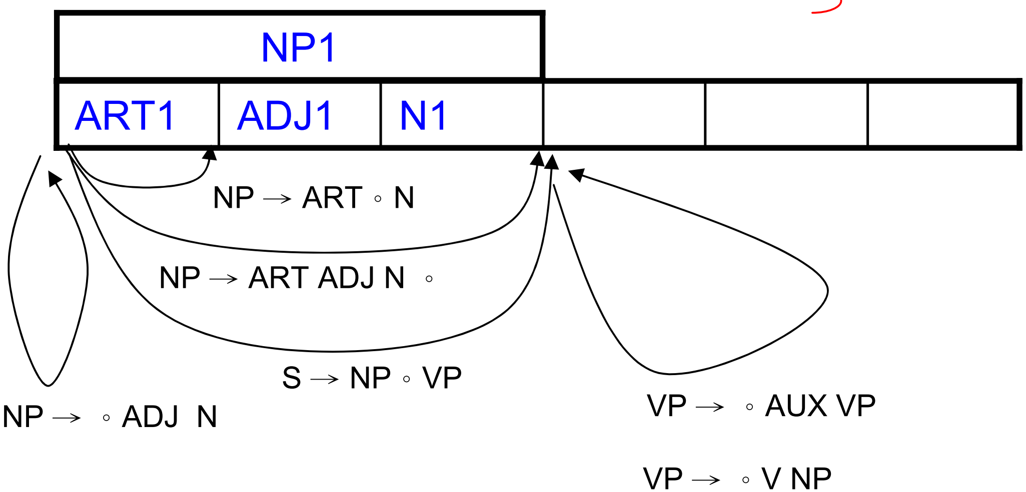
Loop 5

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

- Example

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

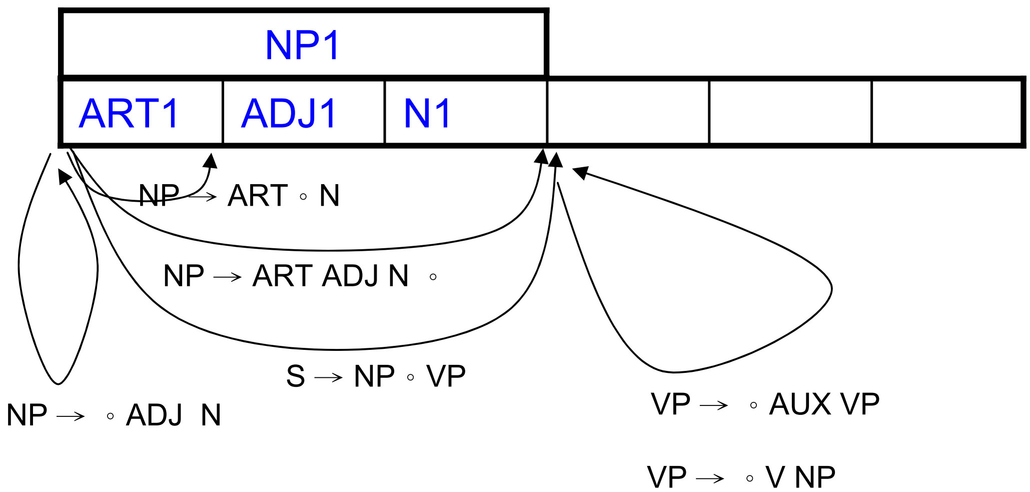
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Chart:



Agenda:

N2 4,5
V1 4,5

The Top-Down Chart Parser

- 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

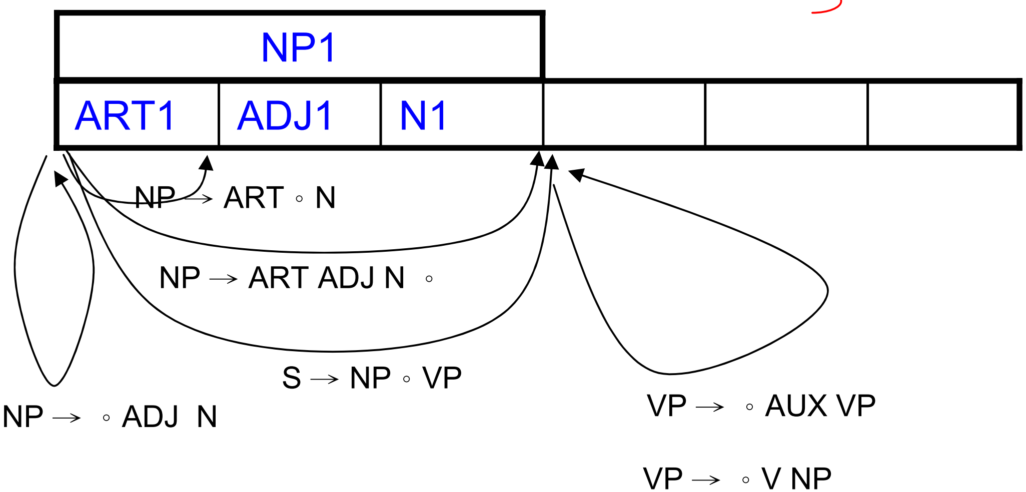
Loop 6

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

- Example

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

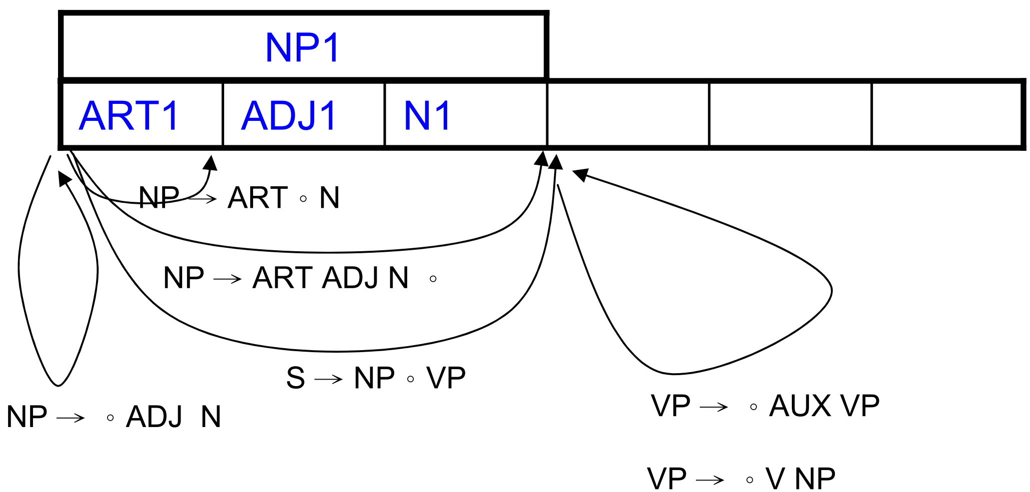
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

- Example

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

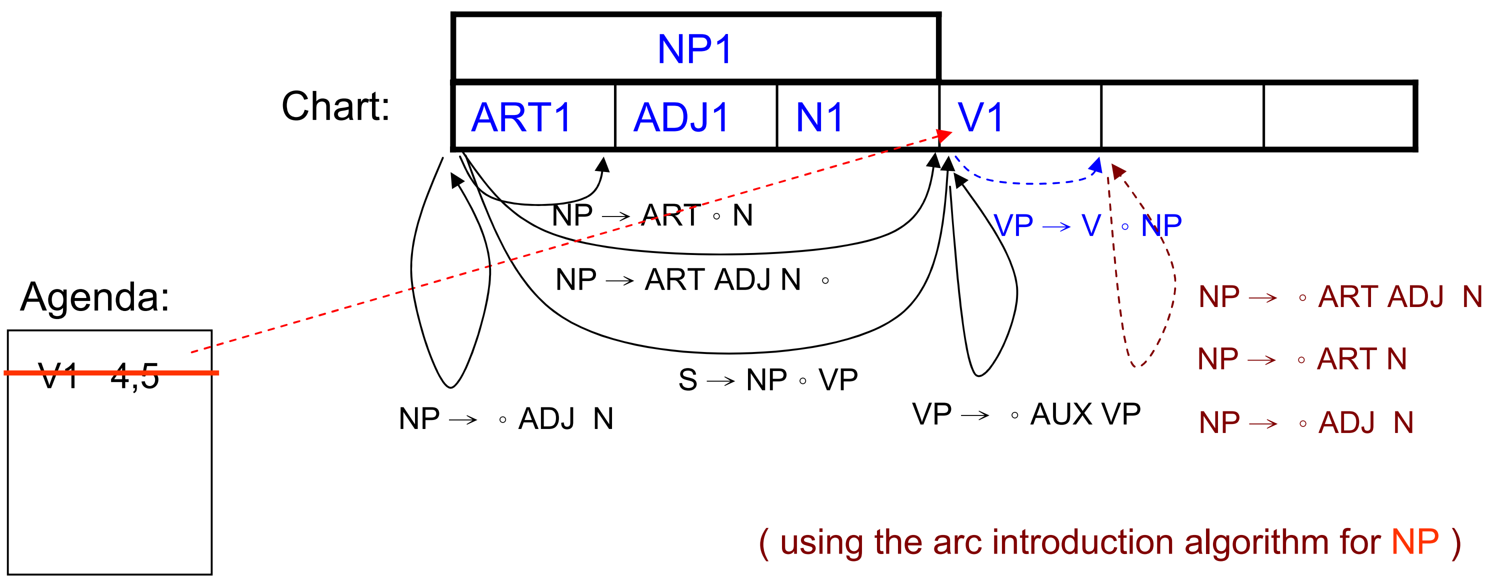
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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)



The Top-Down Chart Parser

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

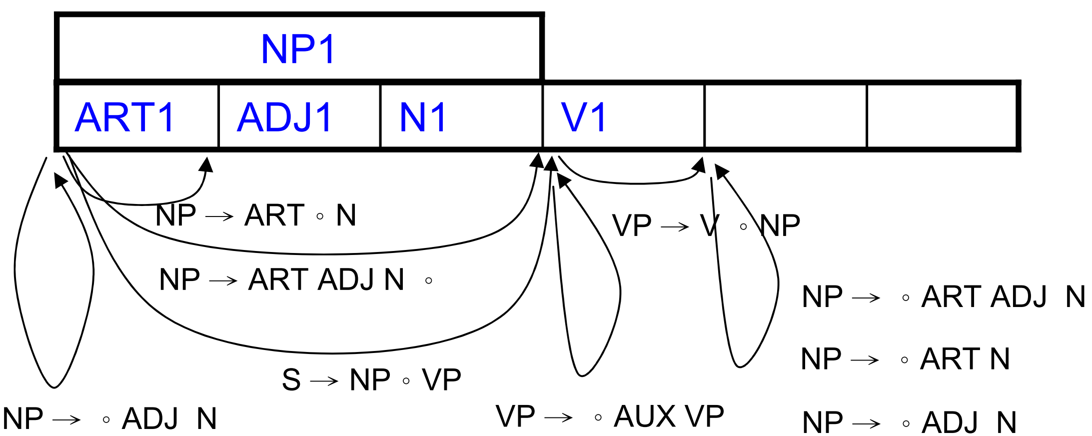
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Loop 8

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

Chart:



Agenda:

ART2 5,6

The Top-Down Chart Parser

- Example

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

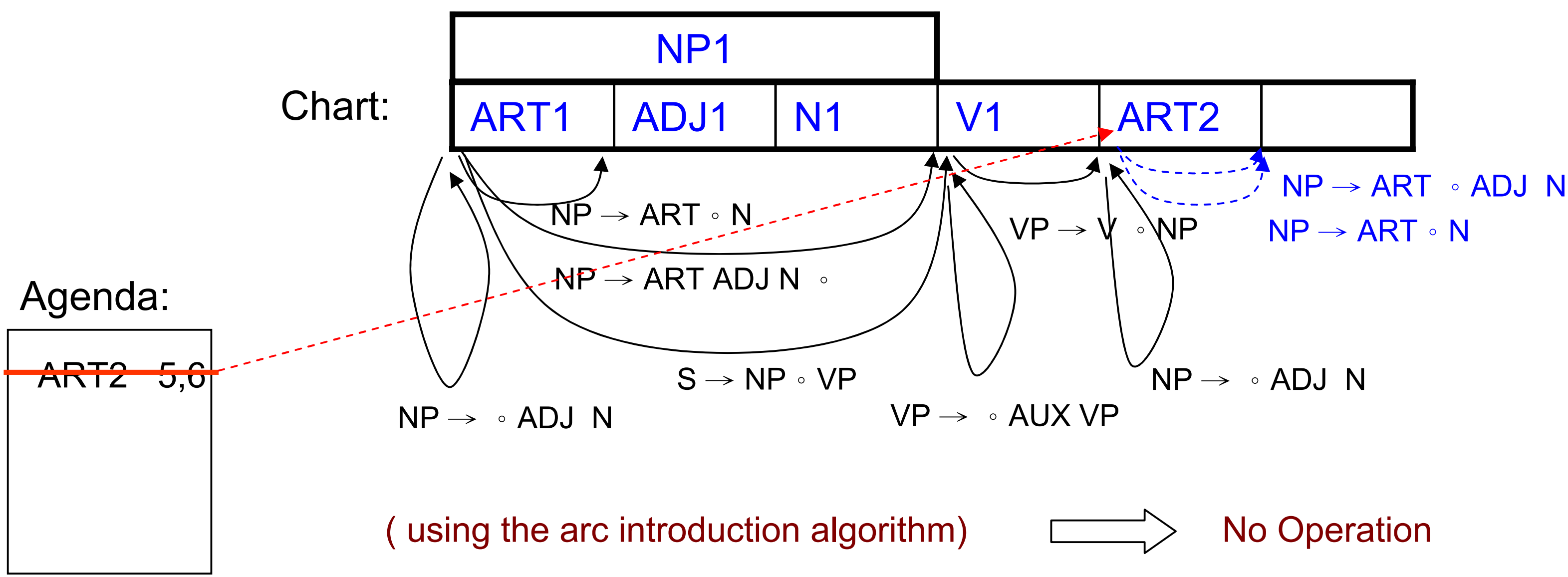
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

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

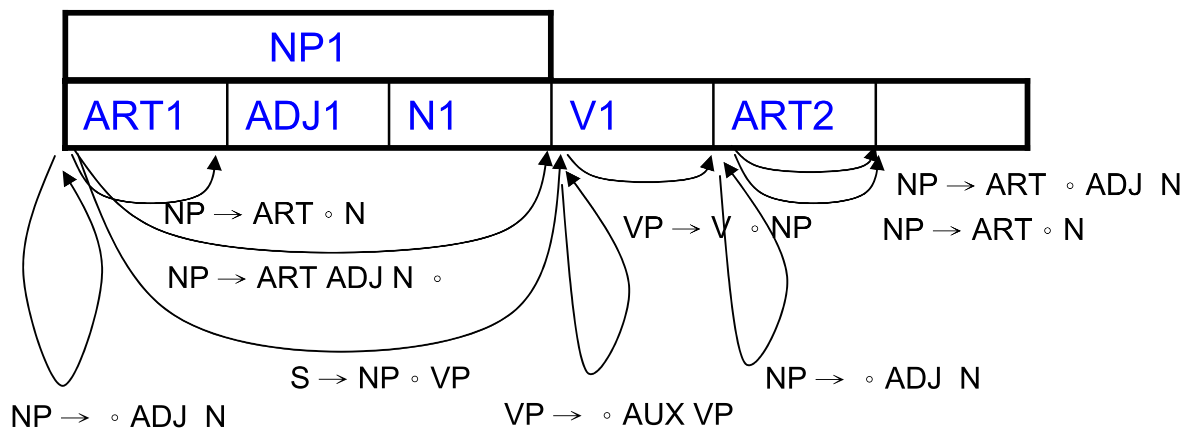
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

- Example

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

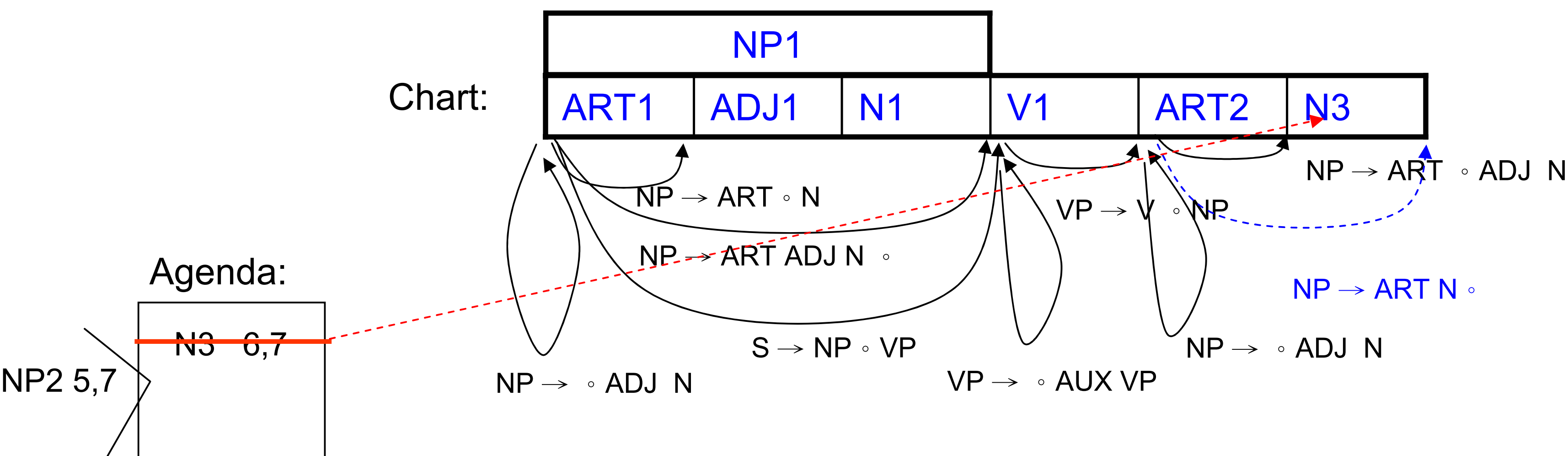
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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)



(using the arc introduction algorithm) ⇨ No Operation

The Top-Down Chart Parser

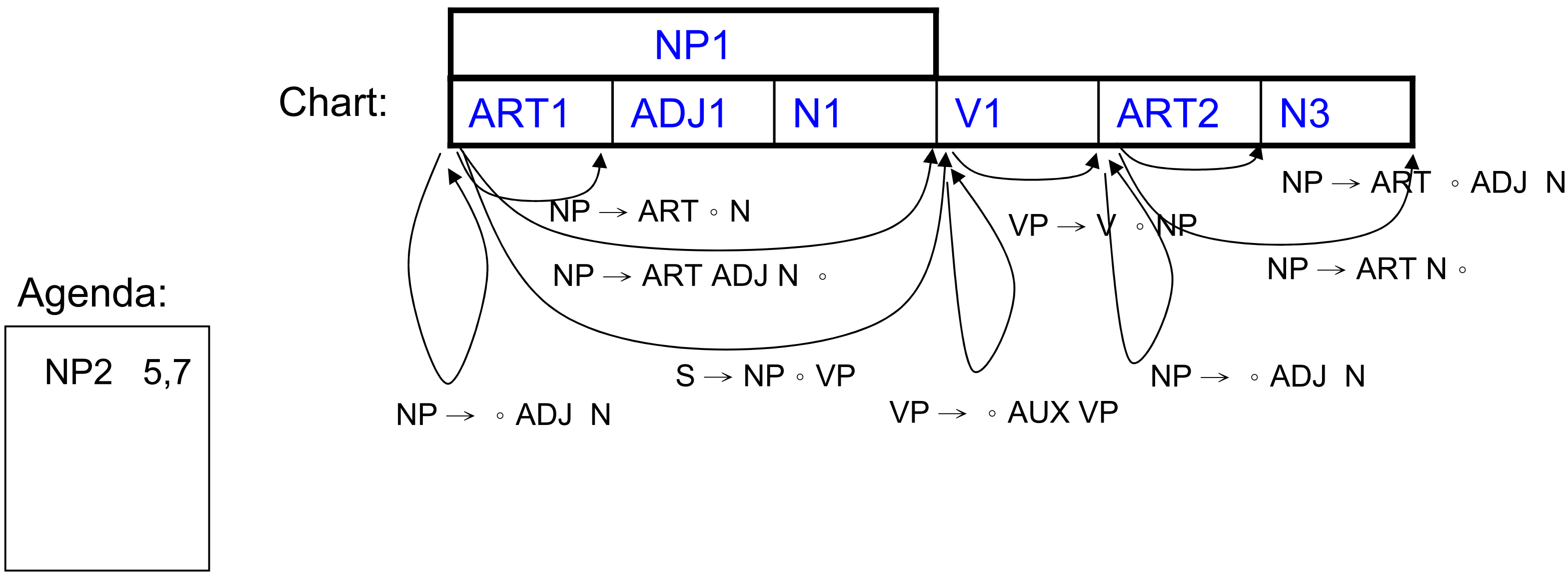
- Example $_1$ The $_2$ large $_3$ can $_4$ holds $_5$ the $_6$ water $_7$

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):



The Top-Down Chart Parser

- 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

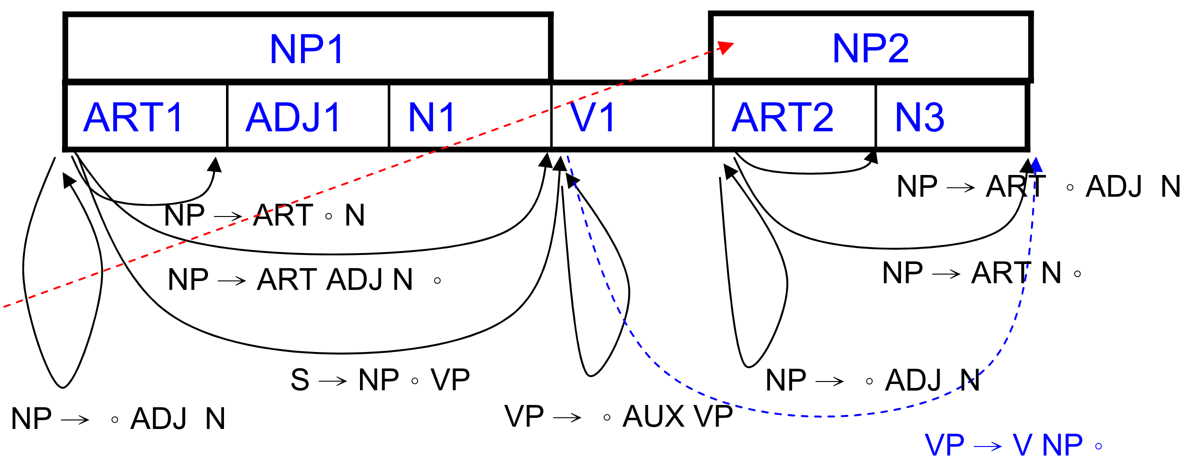
the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

Loop 10

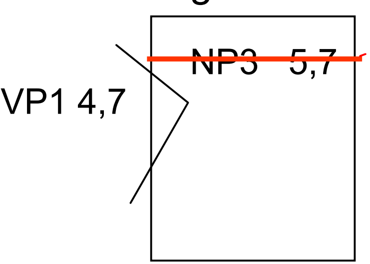
Enter NP2 (*"the water"* from 5 to 7):

(using the arc extension algorithm)

Chart:



Agenda:



(using the arc introduction algorithm) ⇨ No Operation

The Top-Down Chart Parser

- Example

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

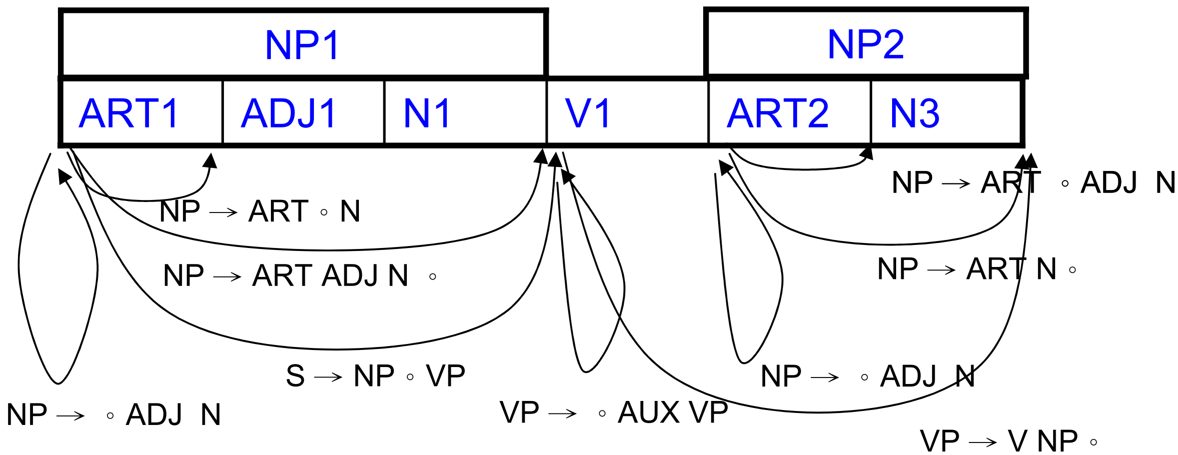
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

Loop 11

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

Chart:



Agenda:

VP1 2 4,7

The Top-Down Chart Parser

- Example

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

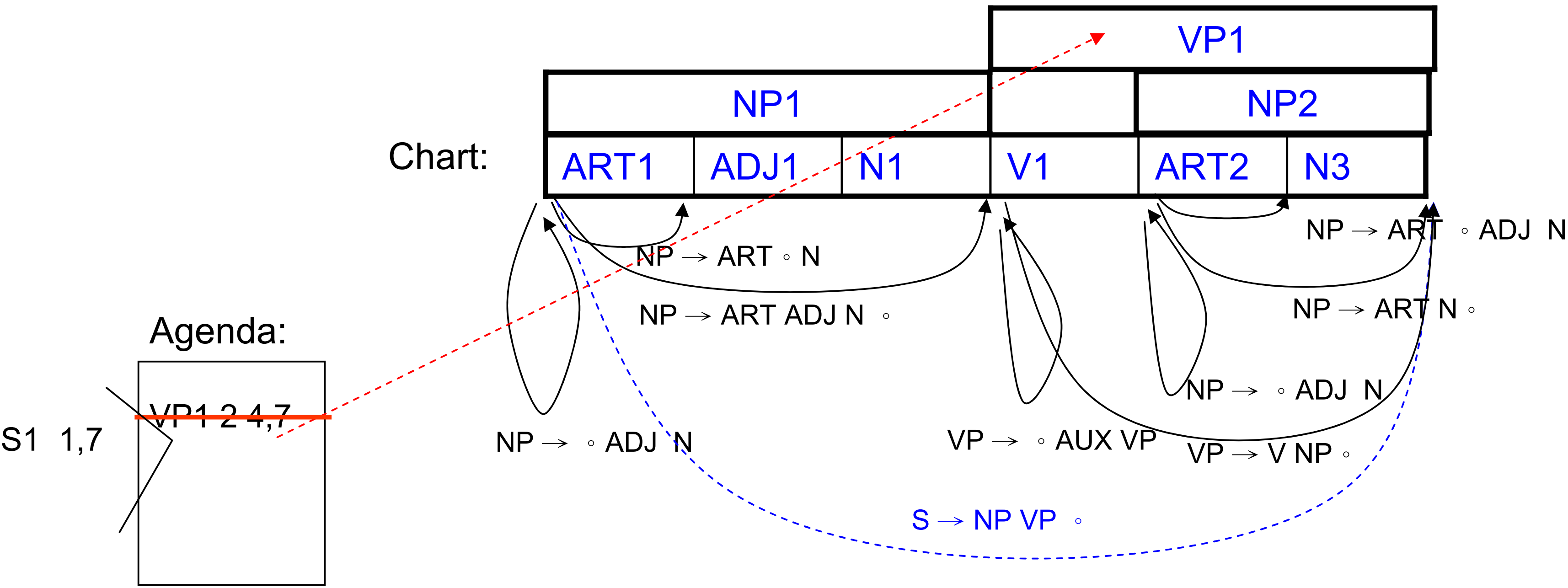
1. S → NP VP	4. NP → ADJ N
2. NP → ART ADJ N	5. VP → AUX VP
3. NP → ART N	6. VP → V NP

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

- 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

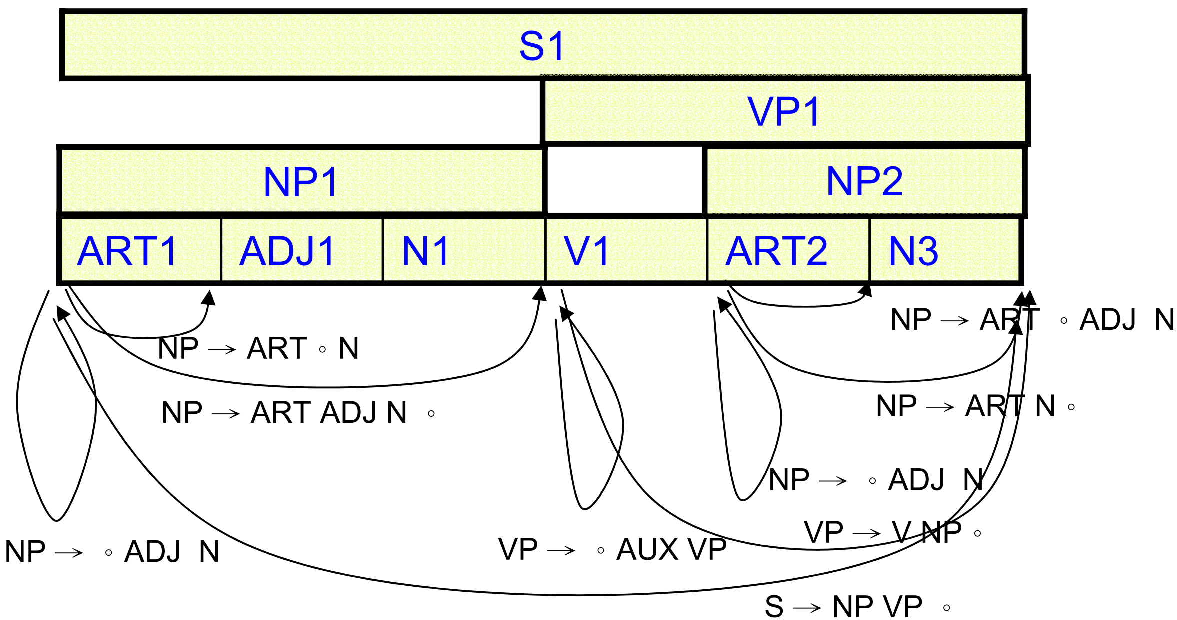
Loop 12

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

Agenda:

S1 1,7

Chart:



The Top-Down Chart Parser

- 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

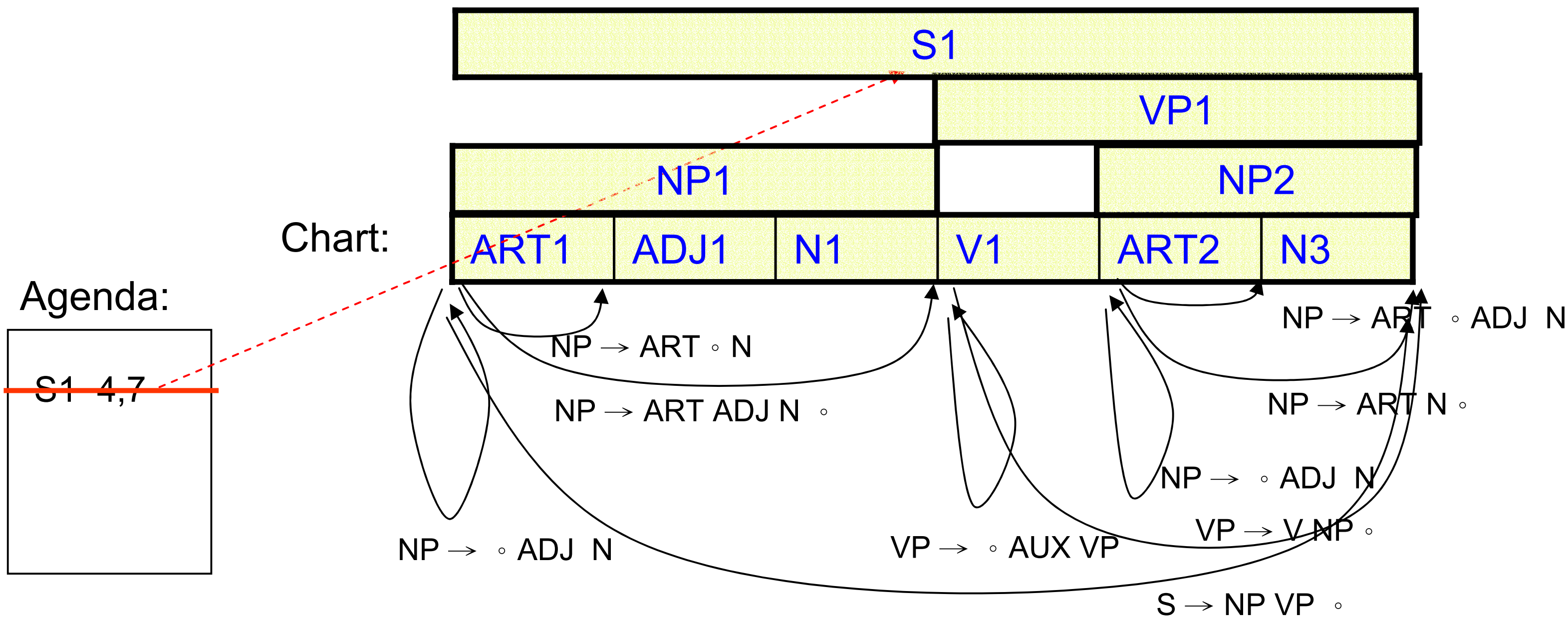
the: ART
large: ADJ
can: N, AUX
hold: N, V
Water: N

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



Comparisons

- 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