Natural Language Parsing with Context-Free Grammars

SPFLODD

Spring 2018

Problem: Extracting meaning or intent from natural language..

- Whole Foods Raises Prices For Suppliers

- What happened?

Extracting meaning or intent from natural language..

- | Whole Foods | Raises Prices | For Suppliers |

- What happened?

Meaning from natural language..

- Asteroid Skimming Past Earth May Loom Larger Than Exploding Russian Meteor

Meaning from natural language..

- *Asteroid Skimming Past Earth May Loom Larger Than Exploding Russian Meteor*

Meaning from natural language..

- Uber and Waymo have reached a $245 million settlement in their massive legal fight over self-driving-car technology

Meaning from natural language..

- **Uber and Waymo** *have reached a $245 million settlement* **in their massive legal fight** *over self-driving-car technology*

Meaning from natural language..

- **Uber and Waymo** ***have reached a $245 million settlement*** **in their massive legal fight** *over self-driving-car technology*

- Sentences are composed of groups of words that carry key elements of their meaning
  - Changing these groups will *locally* modify the meaning..

Problem: Extracting meaning or intent from natural language..

- **Uber and Waymo** ***will eat a banana pie*** **in their massive legal fight** ***over self-driving-car technology***

- Sentences are composed of groups of words that carry key elements of their meaning
  - Changing these groups will *locally* modify the meaning..

Problem: Extracting meaning or intent from natural language..

- **Uber and Waymo** ***have eat a banana pie*** **in their massive legal fight** ***to determine who's a better chimpanzee***

- Sentences are composed of groups of words that carry key elements of their meaning
  - Changing these groups will *locally* modify the meaning..

Problem: Extracting meaning or intent from natural language..

- **The champions of banana** ***will eat a banana pie*** **in their massive legal fight** ***to determine who's a better chimpanzee***

- Sentences are composed of groups of words that carry key elements of their meaning
  - Changing these groups will *locally* modify the meaning..

Problem: Extracting meaning or intent from natural language..

- **The champions of banana** ***will eat a banana pie*** **in a friendly faceoff** ***to determine who's a better chimpanzee***

- Sentences are composed of groups of words that carry key elements of their meaning
  – Changing these groups will *locally* modify the meaning..

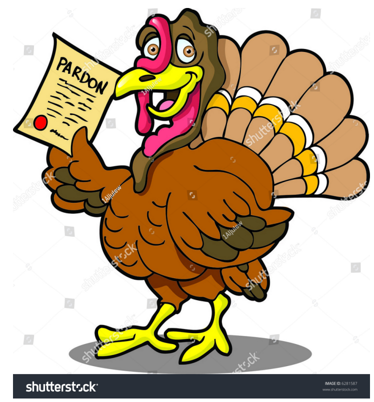
Can be reordered

- **The champions of banana will *eat a banana pie to determine who's a better chimpanzee* in a friendly faceoff**
  - The champions of banana will eat a banana pie to determine who's a better chimpanzee in a friendly faceoff

- **Whole foods raises prices for suppliers**
  - For suppliers whole foods raises prices

Grouping the words differently can change meaning

- Spare him *not kill him*



- Spare him not *kill him*

Though regrouping may sometimes permit Yoda..

- **The champions of banana will** ***eat a banana pie*** **in a friendly faceoff** ***to determine who's a better chimpanzee***

- ***To determine who's a better chimpanzee*** **in a friendly faceoff** ***eat a banana pie*** **the champions of banana will**

Contiguity

- The groups are (generally) continguous
  - **The champions of banana** **will** ***eat a banana pie*** **in a friendly faceoff** ***to determine who's a better chimpanzee***
- The groups often have subgroups
  - **The champions of banana** **will** ***eat a banana pie*** **in a friendly faceoff** ***to determine who's a better chimpanzee***
- But splitting and redistributing the segments can change the meaning (if the result is

Meaning from natural language..

- The theory that sentences in a language have recursive block structure is ~2500 years old!
  - Pāṇini

  - An essential property of these block structures is that logical units never overlap
    - i.e. each block is contiguous
      - Although subblocks may be reordered within a block

Constituents

- **The champions of banana will** ***eat a banana pie*** **in a friendly faceoff** ***to determine who's a better chimpanzee***

- We will call these word groups ***constituent phrases*** of the sentence
  - Constituents may have constituents..

Constituents and parts of speech

- **The champions of banana will** ***eat a banana pie*** **in a friendly faceoff** ***to determine who's a better chimpanzee***
- Constituent phrases will typically act as parts of speech
  - E.g.: The champions of banana: Looks like a noun
  - E.g.: will eat a banana : Looks like a verb

Recap: Parts of speech

- Noun: **Names** of persons, places, things, feelings etc
  - John, cat, car, happiness
- Pronoun: Stands for a noun
  - I, you, we
- Verb: Words that represent **action** or **doing**
  - Go, buy, be
- Adjectives: **Modifiers** for ***noun***
  - Tall, fast, happy

Penn Treebank (Marcus et al., 1993)

- A million words (40K sentences) of *Wall Street Journal* text (late 1980s).
  - This is important to remember!
- Parsed by experts; consensus parse for each sentence was published.
- Attempts to be theory-neutral, probably more accurate to say that it represents its own syntactic theory.
- Many other treebanks now available in other

According to the Penn Tree Bank

1. CC Coordinating conjunction
2. CD Cardinal number
3. DT Determiner
4. EX Existential there
5. FW Foreign word
6. IN Preposition or subordinating conjunction

1. PRP$ Possessive pronoun
2. RB Adverb
3. RBR Adverb, comparative
4. RBS Adverb, superlative
5. RP Particle
6. SYM Symbol
7. TO to

Constituents and parts of speech

- Constituent phrases will typically act as parts of speech
  - E.g.: The champions of banana: Looks like a noun
  - E.g.: will eat a banana : Looks like a verb
- Phrases take the characteristics of the *head* word
  - The word that governs the meaning
  - We label the phrase by the POS category of the

Or More Correctly

- Constituent phrases have constituent phrases
  - “The ***champions*** of banana”: Noun phrase
  - “will ***eat*** banana pie in a friendly face off to determine who’s a better chimpanzee”: Verb phrase
  - Note: Simply focusing on the main two terms gives you most of the meaning

- Sub-phrases
  - (NP

Challenge

- *How* to segment the text
  - How to find the constituent phrases
  - Needed, to interpret it

- To do so, we will first describe a *grammar* for the language
- But first.. lets *formally* define language

What is a formal language

•

- A formal language is a se
together with a set of rul
are formed

- More formally:
  – Let Σ be a set of symbols

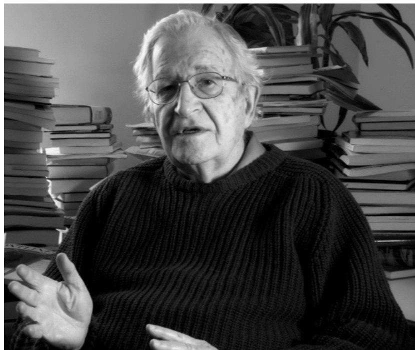
Formal Grammar

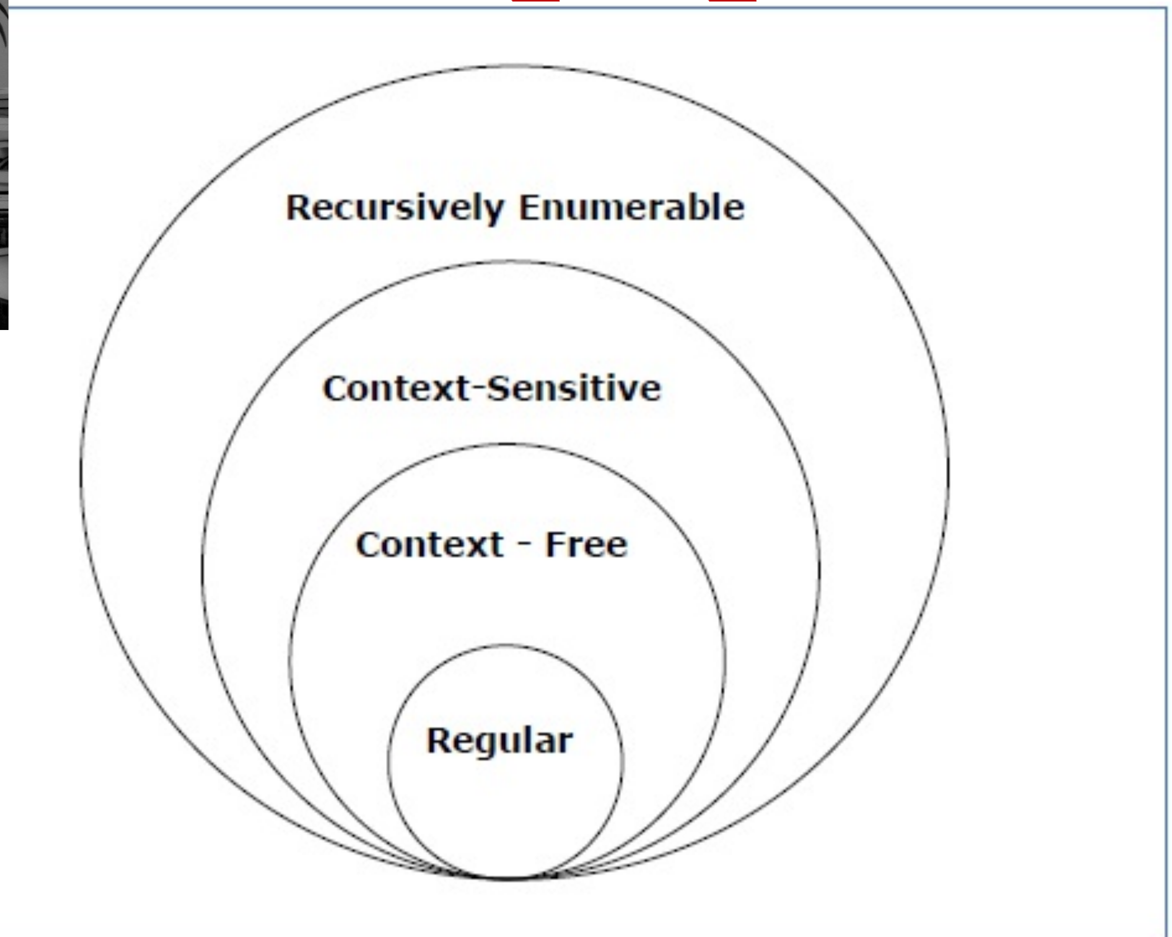
- A formal grammar is the set of language
- It comprises $\langle \Sigma, N, P, S \rangle$:
  - A finite set of *terminal symbo*
    - Also called its alphabet
  - A finite set of *non-terminal* sy
  - A set of *production rules* P

“Recognizing” a language

-
  - Given a language L c
    with a specific set of
    rules..

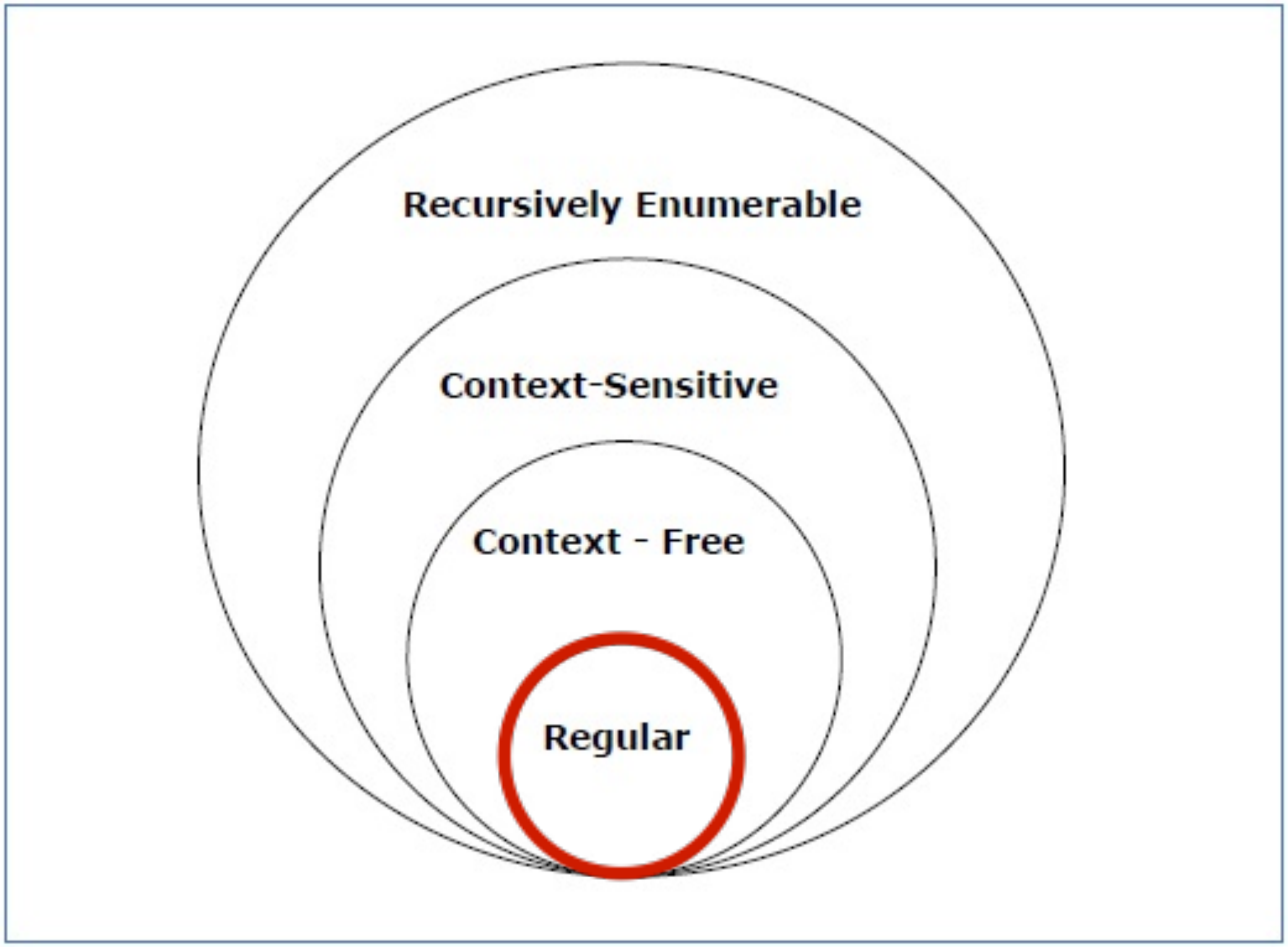
The Chomsky Hierarchy of Language





The Chomsky hierarchy of languages. Each language type is characterized by the type of grammar required to construct it

The Chomsky Hierarchy of Language



The Chomsky hierarchy of languages. Each language type is characterized by the type of grammar required to construct it

Regular language

-
- A *regular* language can be produc
- A regular grammar has rules of th
  - $B \longrightarrow a; \quad B \in N,\ a \in \Sigma$
  - $B \longrightarrow aC; \quad C \in N$
  - $B \longrightarrow \ \varepsilon,$ where ε is the empty st

Regular language

- A *regular* langu

- A regular gram
  - $B \longrightarrow a; \quad B$
  - $B \longrightarrow aC; \quad ($
  - $B \longrightarrow \varepsilon,$ wl

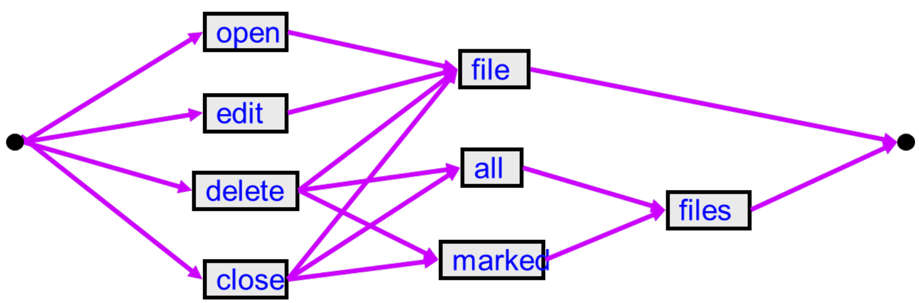
All strings produced by a regular grammar can be viewed as begin obtained by incremental extension of substrings by one symbol at a time

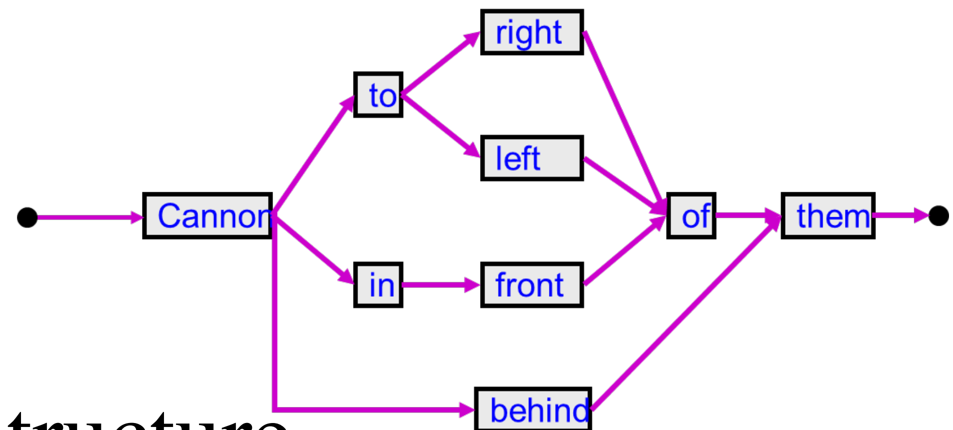
Even if the grammar originally specified increments in blocks of symbols with more complex rules, it can be redefined as an equivalent grammar based on incremental one-symbol extensions

If such redefinition is not possible, the grammar is not regular

Regular language examples

- Regexps
- Restricted sets of commands, poetry..

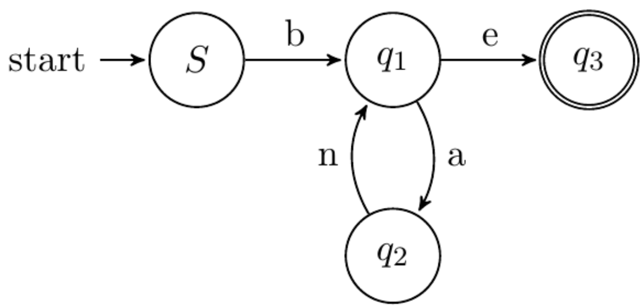




- Or even fairly complex structure
  - Can even produce all possible word sequences for a fixed vocabulary
    - But not “selective” enough for proper natural language

Regular Languages

DF
A

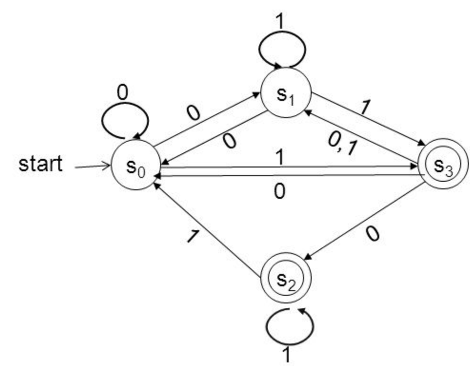


Finite State Automaton, accepting the pattern $b(an)+e$

NDF
A

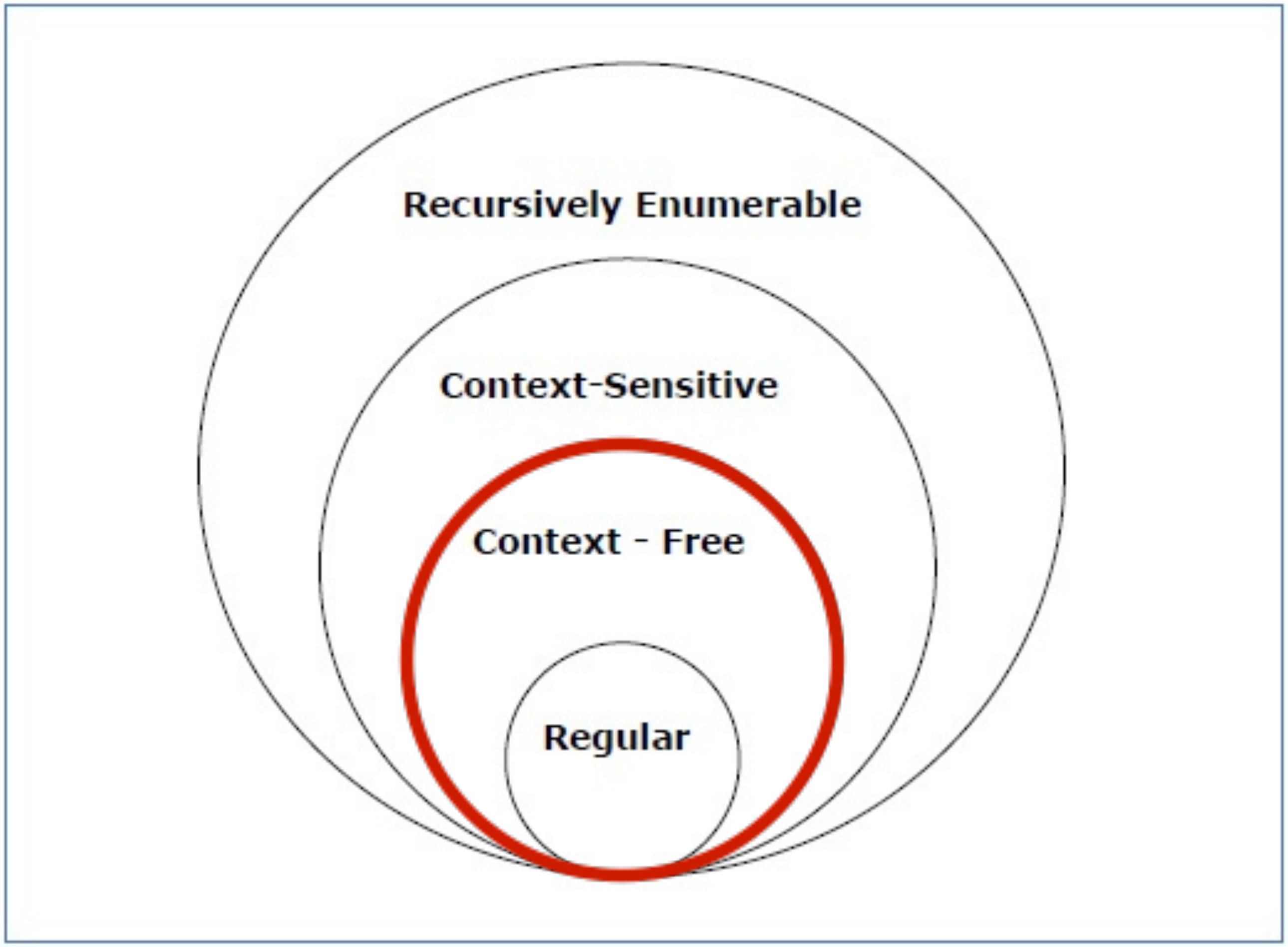
Example : Find the state diagram for the NDFA with the state table shown in table. The final states are $s_2$and s_3

state	f Input 0	1
s_0	s_0 , s_1	s_3
s_1	s_0	s_1, s_3
s_2		s_0, s_2
s_3	s_0, s_1, s_2	s_1



- Regular languages can be recognized by finite-state automatons
  - A machine with a finite number of states, including some "terminal" states
  - Transitions from one state to another by

The Chomsky Hierarchy of Language



The Chomsky hierarchy of languages. Each language type is characterized by the type of grammar required to construct it

CFG

- A CFG comprises $\langle \Sigma, N, P, .$
  have the form:
  - $B \longrightarrow something$; $B \in N$
  - Only restriction, the prod
    context it appears in
    - i.e. the LHS of the producti

CFGs are not (necessarily) finite state

- Consider the production rules
  - $S \rightarrow A; \;\; A \rightarrow bAc; \quad A \rightarrow a; \quad A \rightarrow \varepsilon$

- This produces the following strings
  - $\varepsilon\ (empty\ string),\;\; a,\;\; bc,\;\; bac,\;\; bbcc,$
  - Number of bs is equal to the number of
    - May be infinite

- To produce the last c, must be aware

CFGs are not (necessarily) finite state

- Consider the production rules
  - $A \rightarrow bAc; \quad A \rightarrow a; \quad A \rightarrow \varepsilon$
- This produces the following strings
  - ε (*empty string*), $a, bc, bac, bbcc,$
  - Number of bs is equ

    Is this Markov?
    - May be infinite
- To produce the last c, must be aware

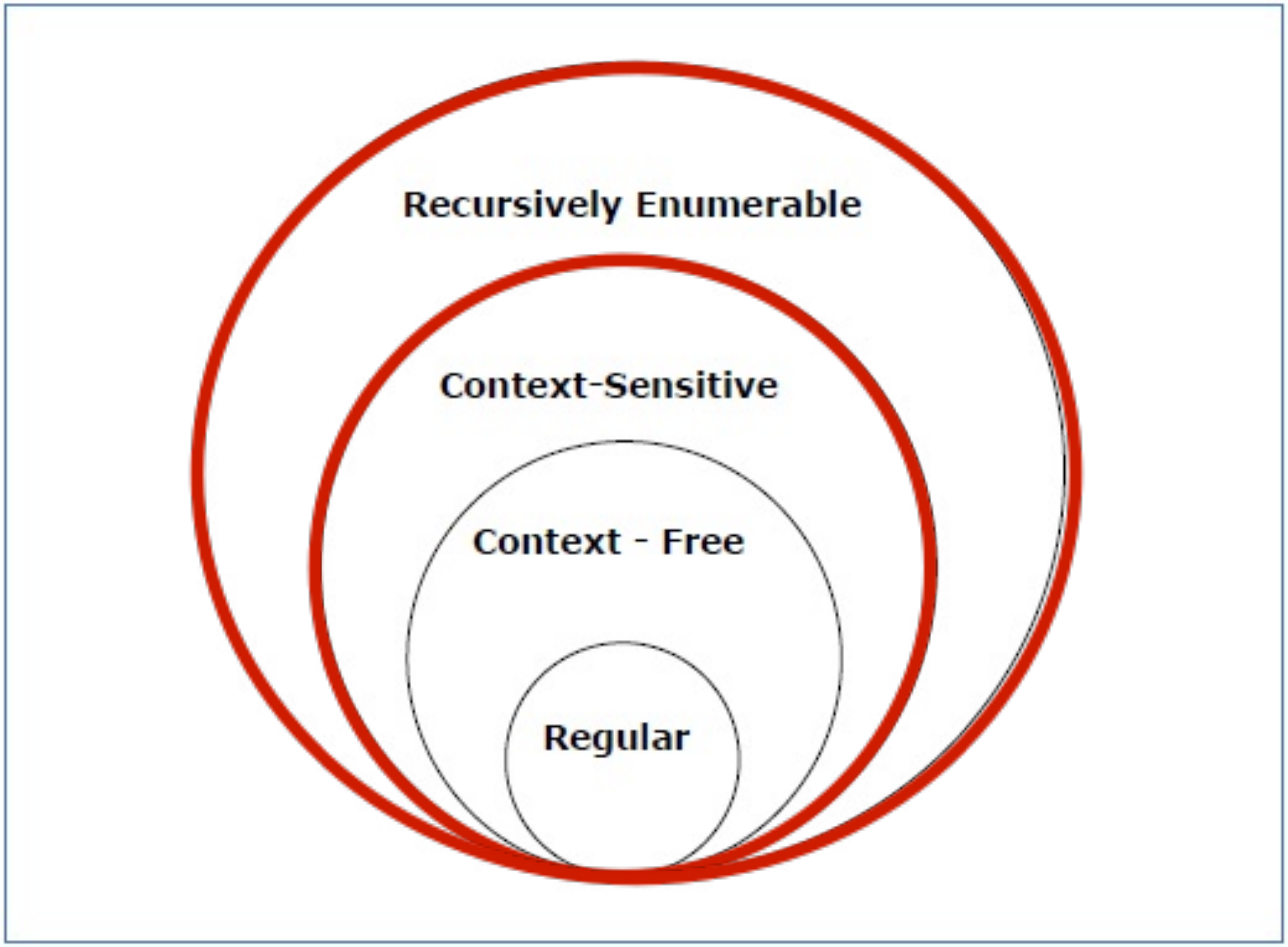
The Chomsky Normal Form

-
  - The CNF representation of a C
  - $G = \langle \Sigma, N, P, S \rangle$
    - $S \rightarrow \varepsilon$ [empty string]
    - $A \longrightarrow a; \quad A \in N, \ a \in \Sigma$
      - Non terminal produces a termi
    - $A \longrightarrow BC; \quad A, B, C \in N$

Examples of languages produced by a CFG

- Programming languages
  - Every open loop must be closed
    - Regardless of size of program within the loop
      - Which may itself have loops
  - Parentheses must be closed
- Natural language?

The Chomsky Hierarchy of Language



The Chomsky hierarchy of languages. Each language type is characterized by the type of grammar required to construct it

Context-sensitive and higher grammars

- Context-sensitive grammar: Production rules depend on context
- Recursively-enumerable grammar: Any grammar that can be recognized by a Turing machine

- Actual unlimited natural language is not well modeled even by recursively enumerable grammar according to Chomsky

Typical uses of language

- *Check* if a given string belongs to a language
  – Can this have been produced by language X
  – Recognition (regexp)
  – Verification (computer programs)

- Guess *how* it was produced
  – Determine its structure
    - *Parsing*

Returning to our problem

- **The champions of banana will** ***eat a banana pie*** **in a friendly faceoff** ***to determine who's a better chimpanzee***

- How do we identify the constituent phrases?

Natural language can be modeled by a CFG

- Grammatically spoken/written language largely follows a CFG structure
  - Natural spoken language doesn't, but parts of it nevertheless do

- Model the language with a CFG
- Determine the constituents of any sentence by *parsing* it with a CFG
  - Its not perfect it's a *model*

An example of a CFG

- $\Sigma = \{that, this, a, the, man, book$
- $N = \{S, NP, NOM, VP, Det, Noun$
- P:
  - $S \rightarrow NP\ VP$
  - $S \rightarrow Aux\ NP\ VP$
  - $S \rightarrow VP$

An example of a CFG

- $\Sigma = \{that, this, a, the, man, book$
- $N = \{S, NP, NOM, VP, Det, Noun$
- P:
  - $S \rightarrow NP\ VP$
  - $S \rightarrow Aux\ NP\ VP$
  - $S \rightarrow VP$

$Det \rightarrow that$

$Noun \rightarrow book \mid f$

$Verb \rightarrow book \mid$

Aux

Note: a non-terminal may be expanded by multiple production rules.
a terminal may appear against multiple non-terminals.

Simplified

- $\Sigma = \{that, this, a, the, man, boo$
- $N = \{S, NP, VP, Det, Noun, Verb$
- P:
  - $S \rightarrow NP\ VP$
  - $S \rightarrow Aux\ NP\ VP$
  - $S \rightarrow VP$

Finding the constituents

- *The man read this book*

$S \to NP\ VP$
$S \to Aux\ NP\ VP$
$S \to VP$
$NP \to Det\ Noun$
$VP \to Verb$
$VP \to Verb\ NP$

$Det \to that \mid this \mid a \mid the$
$Noun \to book \mid flight \mid meal \mid man$
$Verb \to book \mid include \mid read$
$Aux \to does$

S (The man read this book)

Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Det\ Noun$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$
$Noun \rightarrow book \mid flight \mid meal \mid man$
$Verb \rightarrow book \mid include \mid read$
$Aux \rightarrow does$

S (The man read this book)

NP (The man)

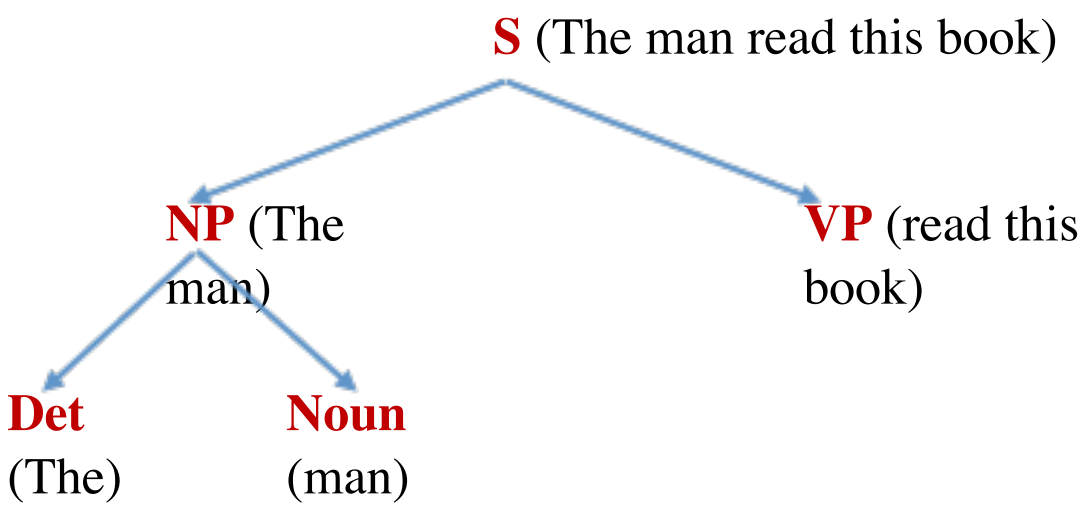
VP (read this book)

Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Det\ Noun$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$
$Noun \rightarrow book \mid flight \mid meal \mid man$
$Verb \rightarrow book \mid include \mid read$
$Aux \rightarrow does$



Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$

$S \rightarrow Aux\ NP\ VP$

$S \rightarrow VP$

$NP \rightarrow Det\ Noun$

$VP \rightarrow Verb$

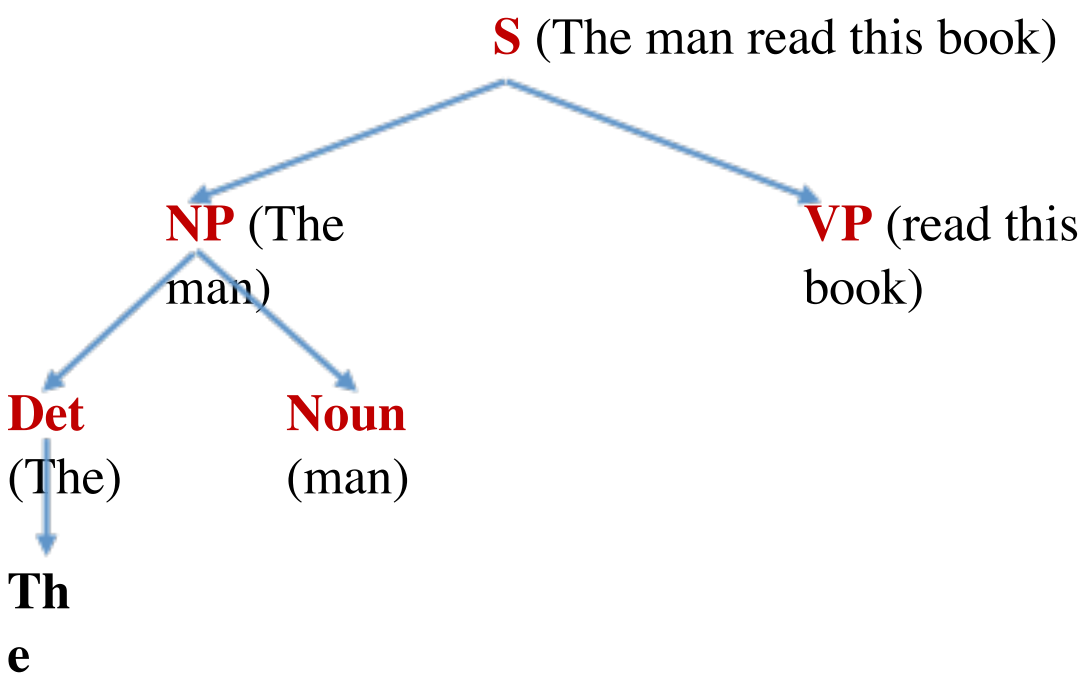
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$

$Noun \rightarrow book \mid flight \mid meal \mid man$

$Verb \rightarrow book \mid include \mid read$

$Aux \rightarrow does$

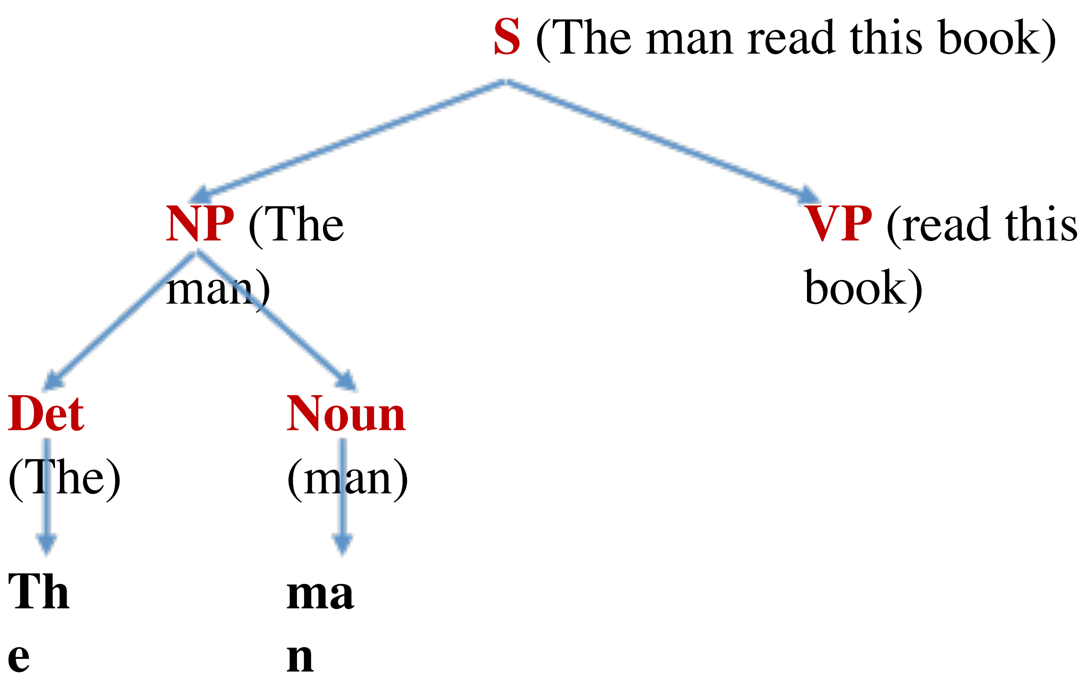


Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Det\ Noun$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$
$Noun \rightarrow book \mid flight \mid meal \mid man$
$Verb \rightarrow book \mid include \mid read$
$Aux \rightarrow does$



Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$

$S \rightarrow Aux\ NP\ VP$

$S \rightarrow VP$

$NP \rightarrow Det\ Noun$

$VP \rightarrow Verb$

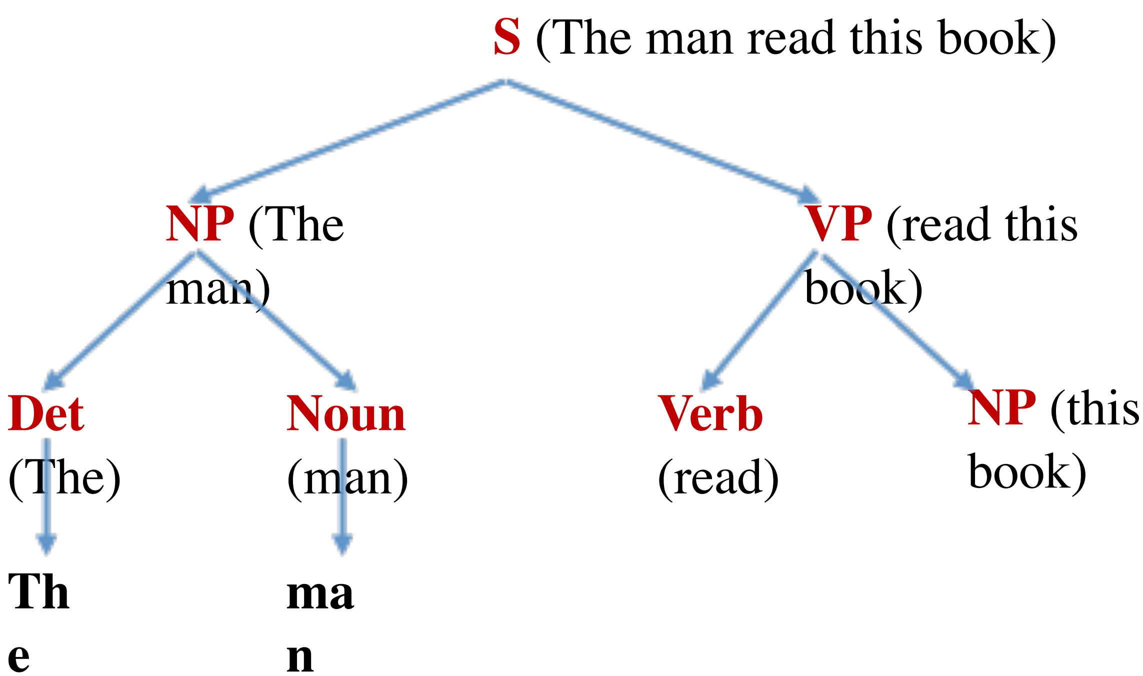
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$

$Noun \rightarrow book \mid flight \mid meal \mid man$

$Verb \rightarrow book \mid include \mid read$

$Aux \rightarrow does$

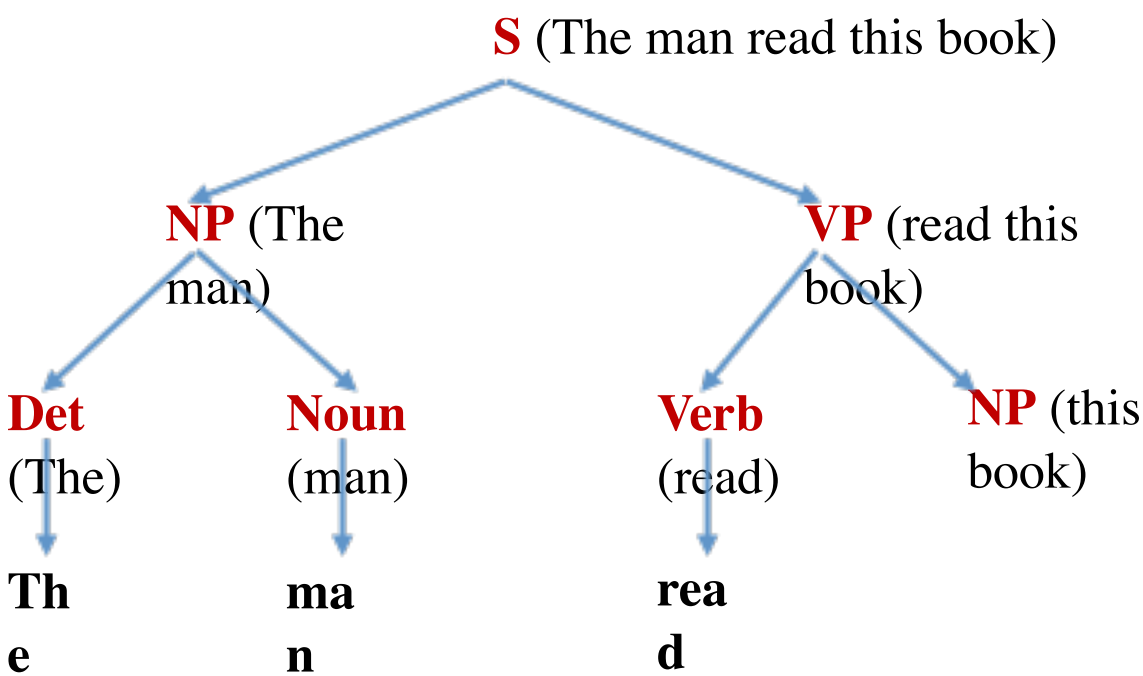


Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Det\ Noun$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$
$Noun \rightarrow book \mid flight \mid meal \mid man$
$Verb \rightarrow book \mid include \mid read$
$Aux \rightarrow does$

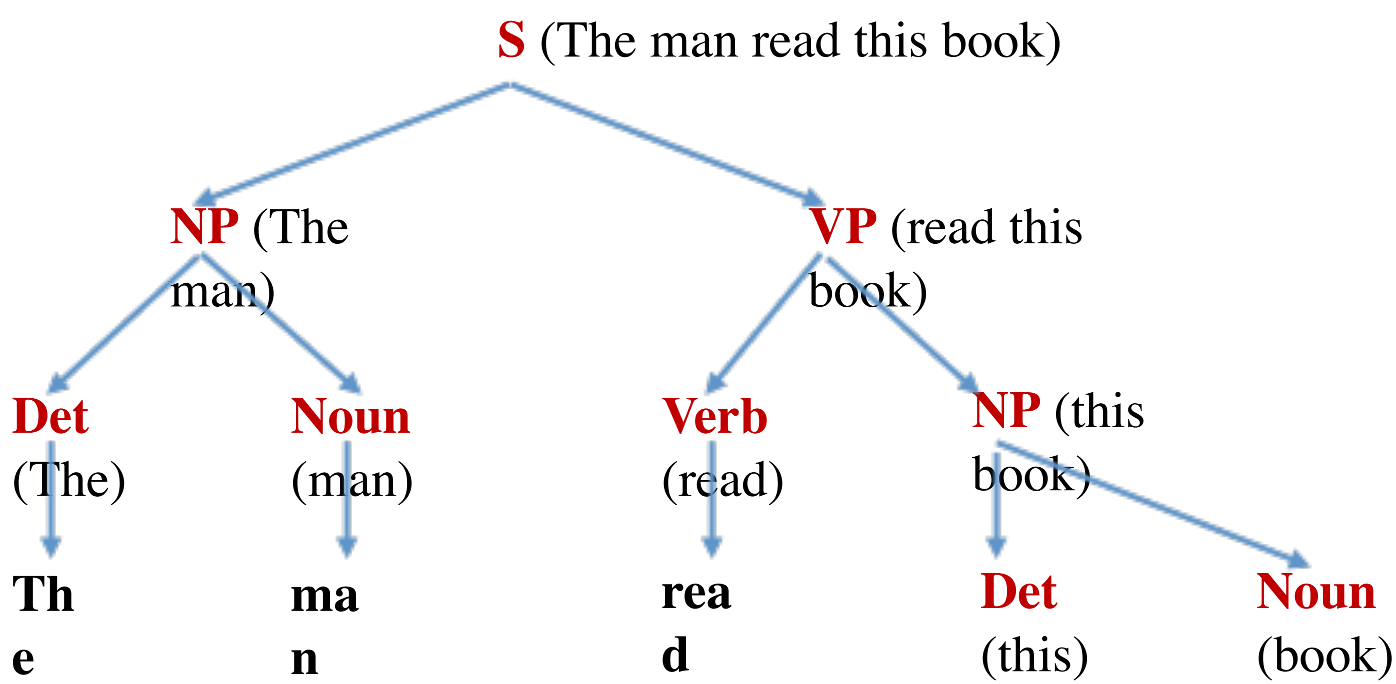


Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Det\ Noun$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$
$Noun \rightarrow book \mid flight \mid meal \mid man$
$Verb \rightarrow book \mid include \mid read$
$Aux \rightarrow does$

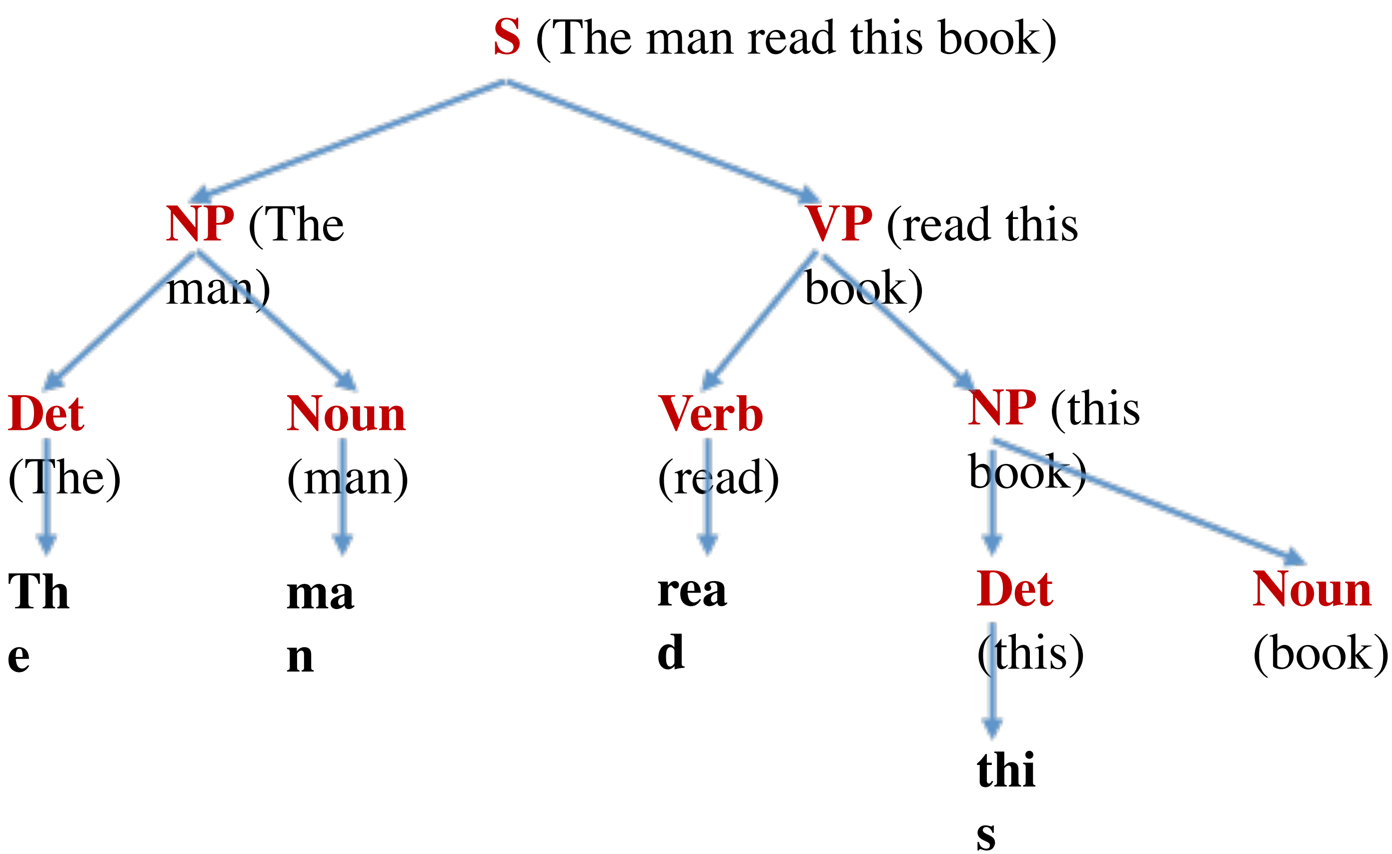


Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Det\ Noun$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$
$Noun \rightarrow book \mid flight \mid meal \mid man$
$Verb \rightarrow book \mid include \mid read$
$Aux \rightarrow does$

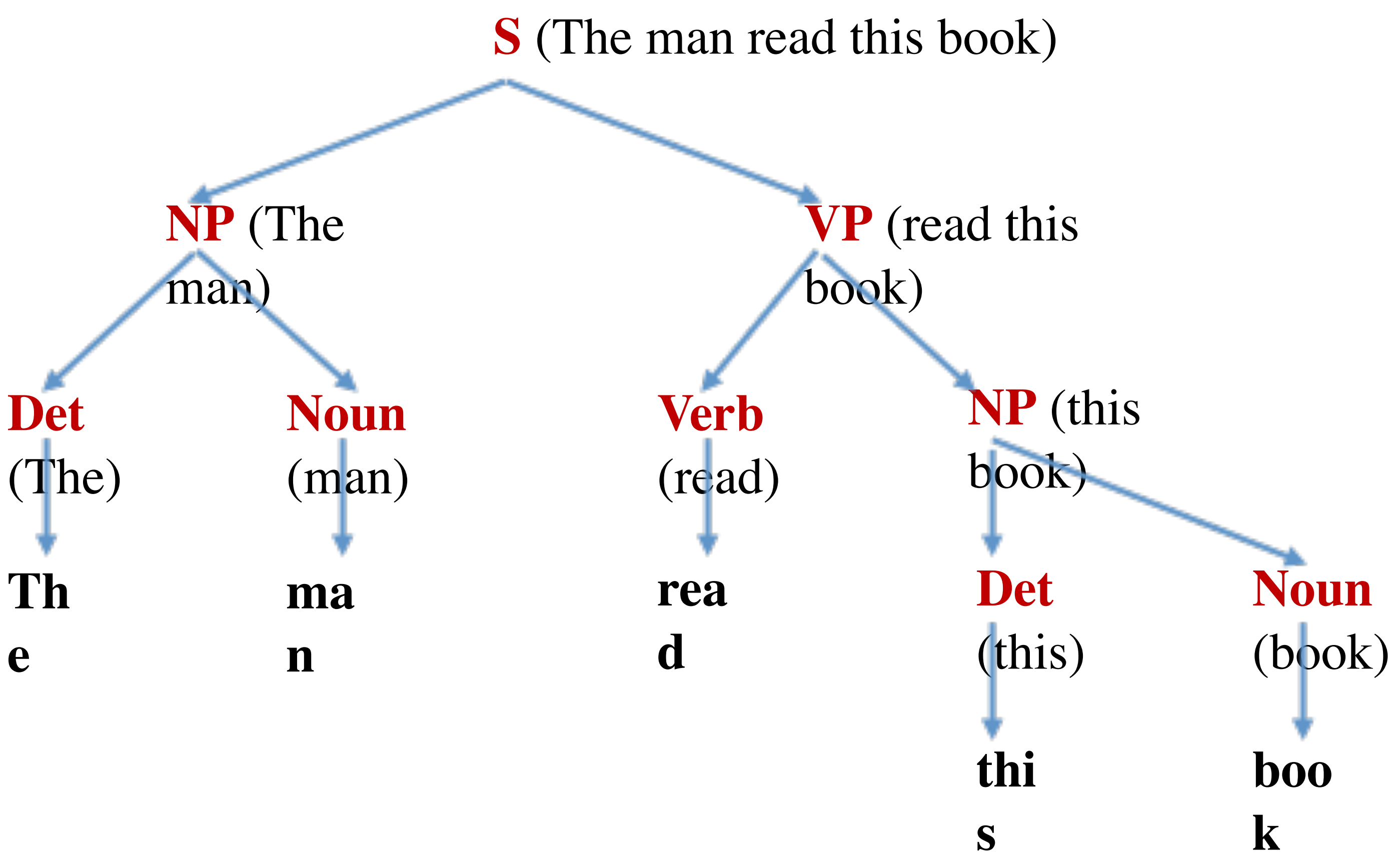


Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Det\ Noun$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$
$Noun \rightarrow book \mid flight \mid meal \mid man$
$Verb \rightarrow book \mid include \mid read$
$Aux \rightarrow does$

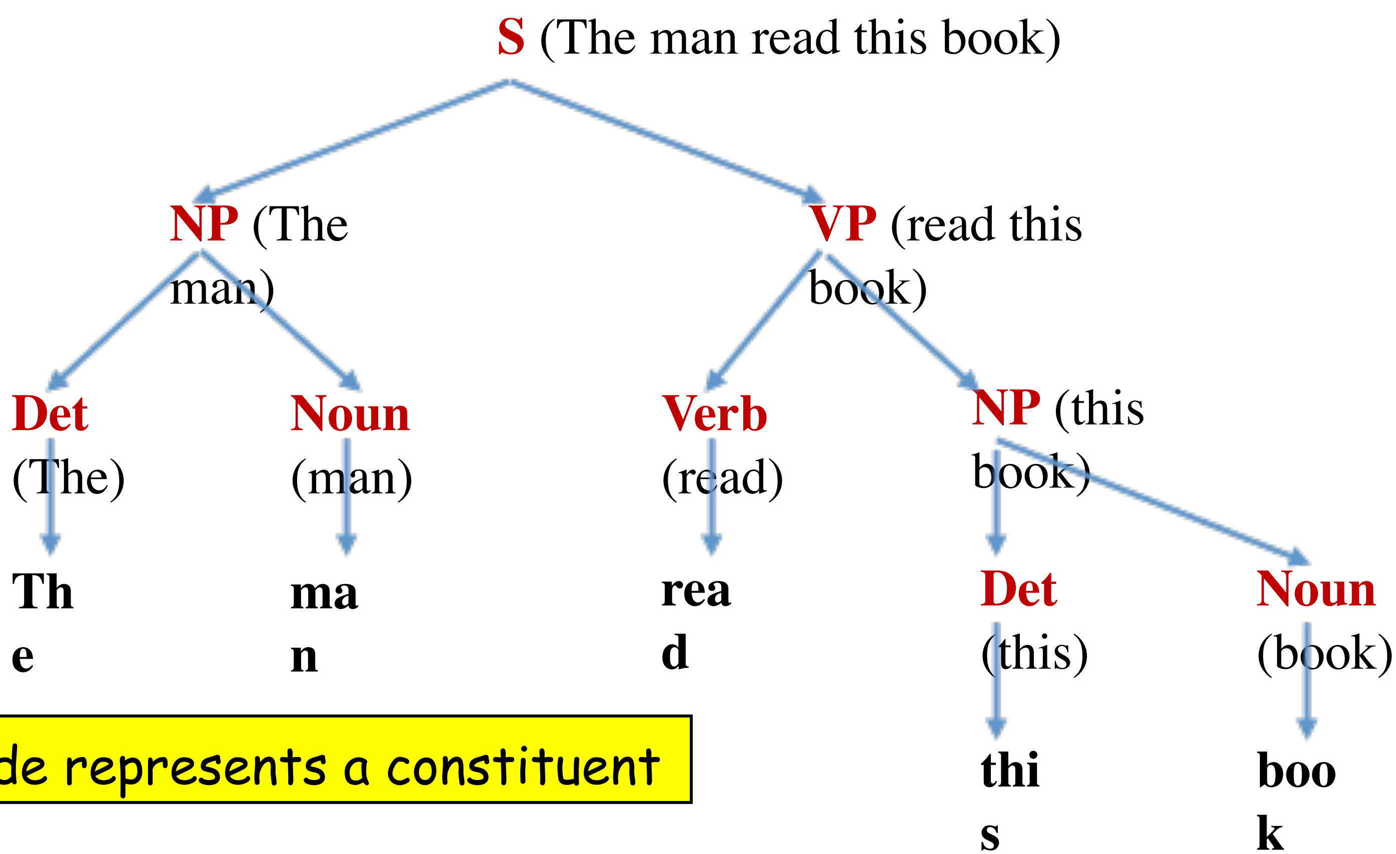


Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Det\ Noun$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$
$Noun \rightarrow book \mid flight \mid meal \mid man$
$Verb \rightarrow book \mid include \mid read$
$Aux \rightarrow does$



Each node represents a constituent

Finding the constituents

- *The man read this book*

$S \rightarrow NP\ VP$
$S \rightarrow Aux\ NP\ VP$
$S \rightarrow VP$
$NP \rightarrow Det\ Noun$
$VP \rightarrow Verb$
$VP \rightarrow Verb\ NP$

$Det \rightarrow that \mid this \mid a \mid the$
$Noun \rightarrow book \mid flight \mid meal \mid man$
$Verb \rightarrow book \mid include \mid read$
$Aux \rightarrow does$

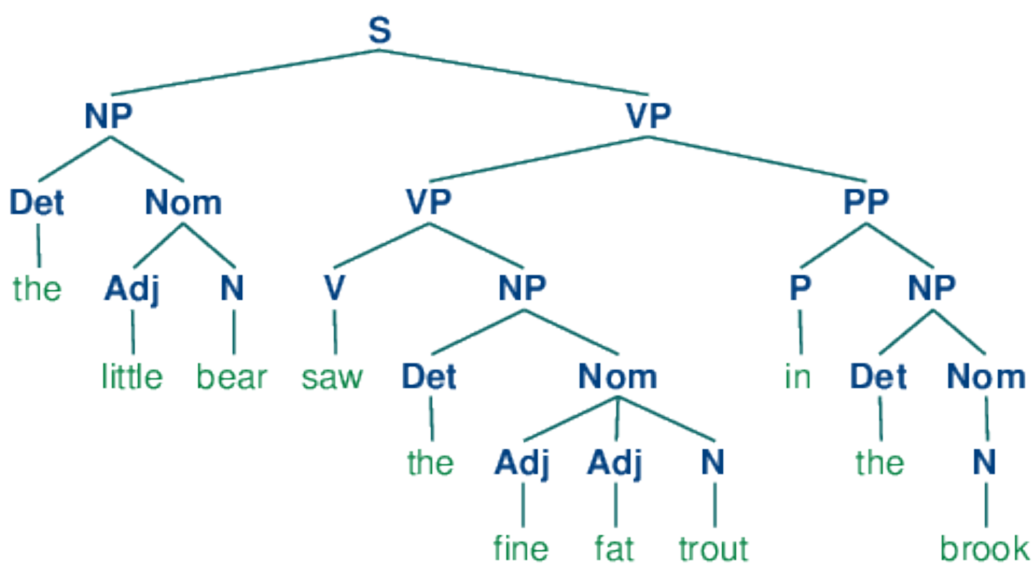
S (The man read this book)

NP (The man)

VP (read this book)

- There are multiple rules expanding **S**. How did we know which one to apply?
  – In general there may be many production rules for any non-terminal. How do we know which one to apply?

Finding the constituents



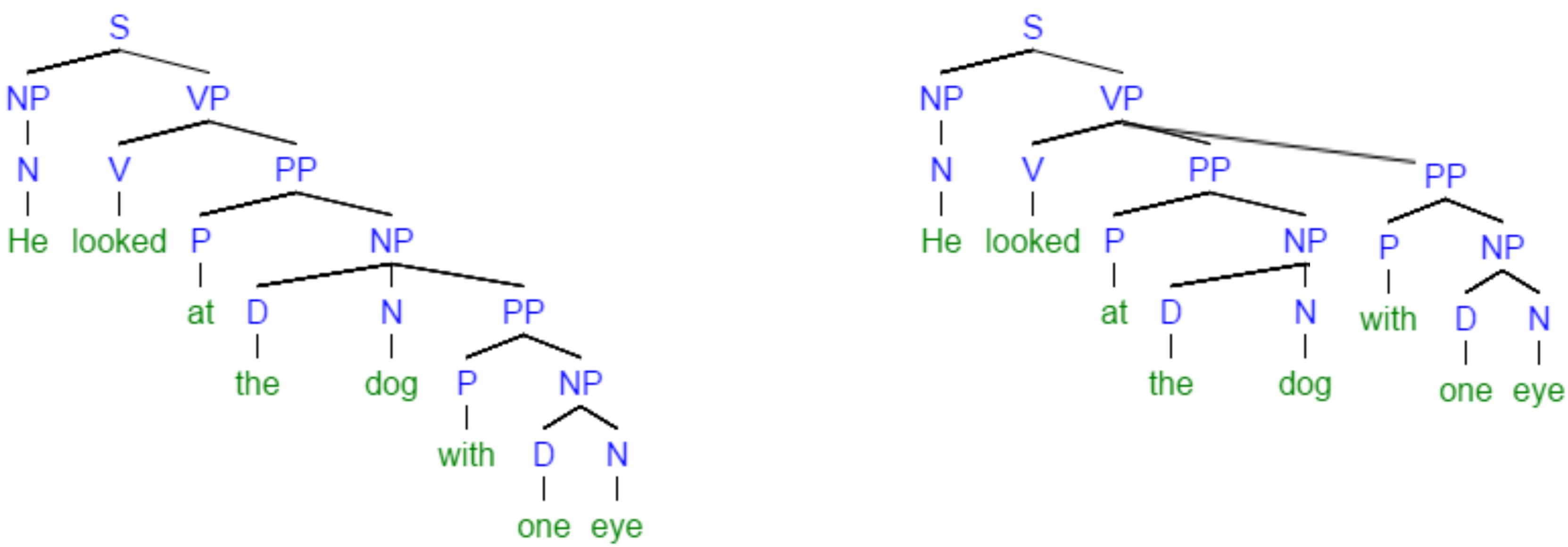
Trout in the brook? 

Saw the fat? 

The little bear saw? 

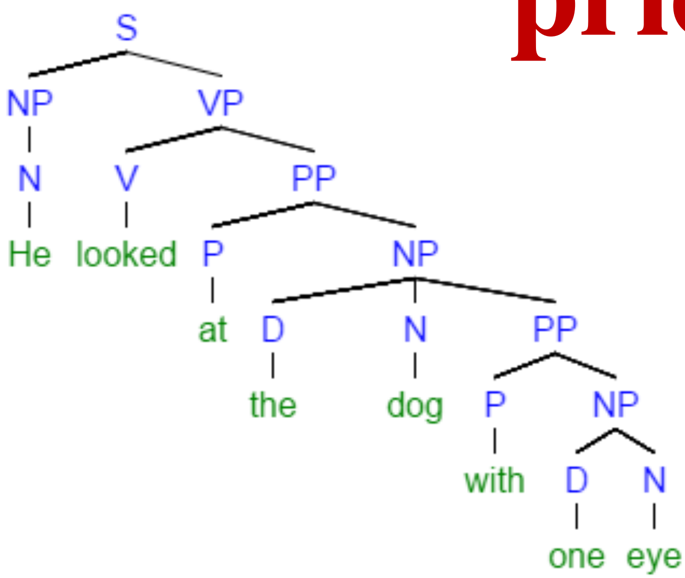
- Finding the right combination to compose the sentence is a challenging search problem
  - **The problem of *parsing***
- ***But wait… it gets tougher..***

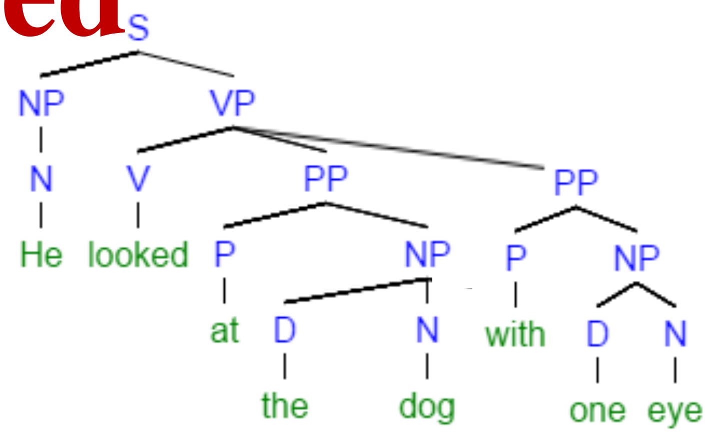
Parses are not unique



- Parses can be ambiguous
  - Grammars can be ambiguous
    - Admit multiple parses
  - English is an ambiguous language

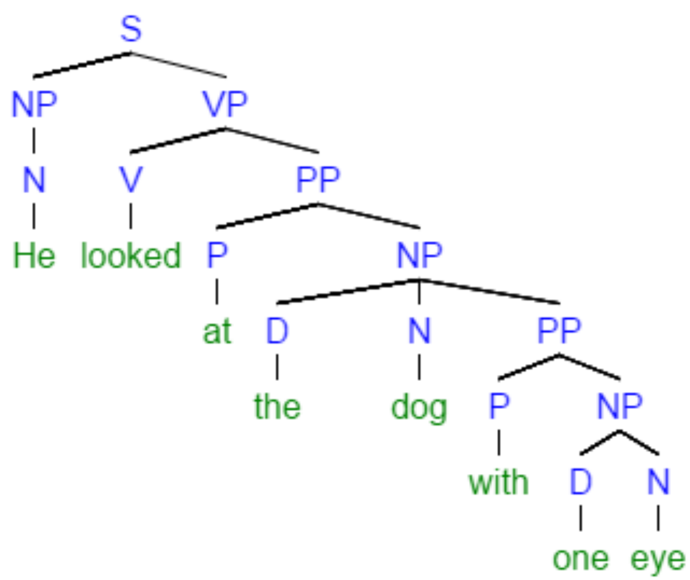
Problem: Production rules are not prioritized

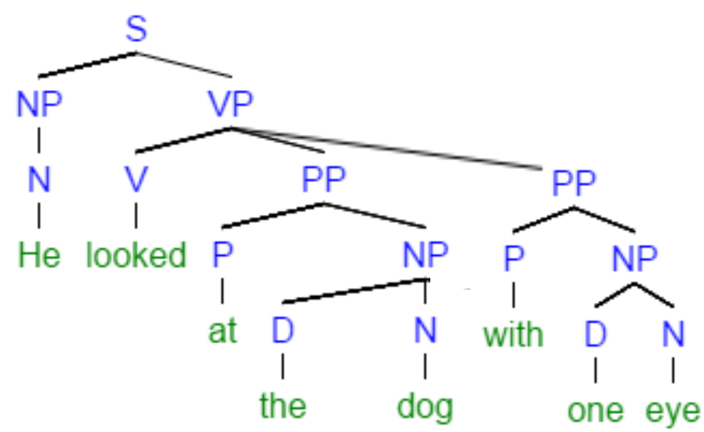




•
- Consider this (not so great) exa
  - $VP \rightarrow V\ PP$
  - $VP \rightarrow V\ PP\ PP$

Disambiguating: Attempt 1

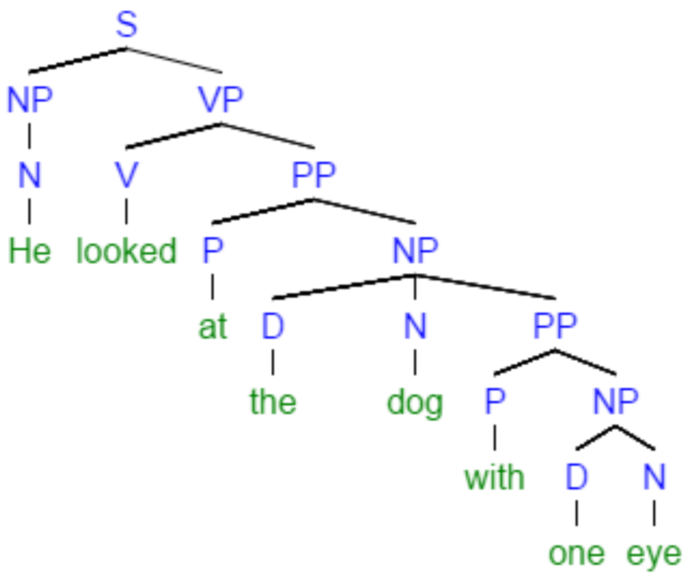


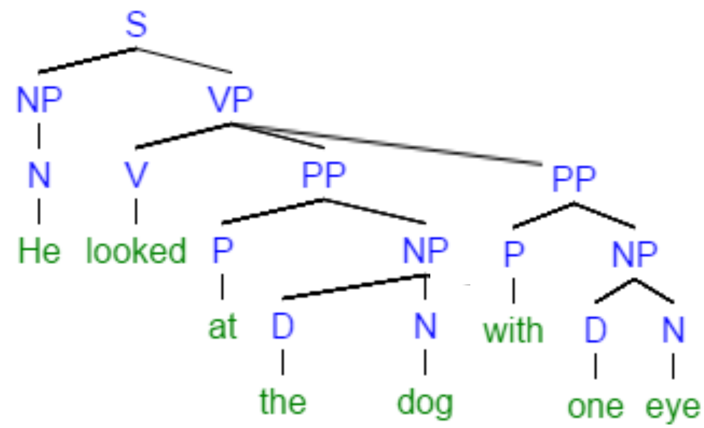


•

- Probabilistic selection:
  - $P(parse) = f(f_1(tree), f_2$
    - E.g. $P(parse) \propto exp(\sum_i \lambda_i$

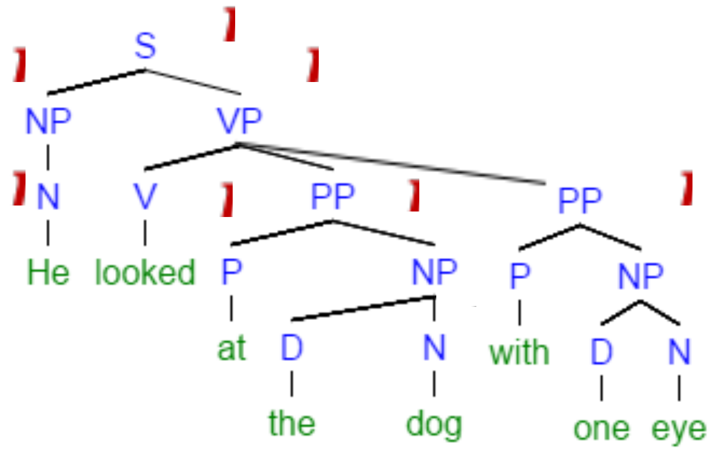
Disambiguating: Attempt 1





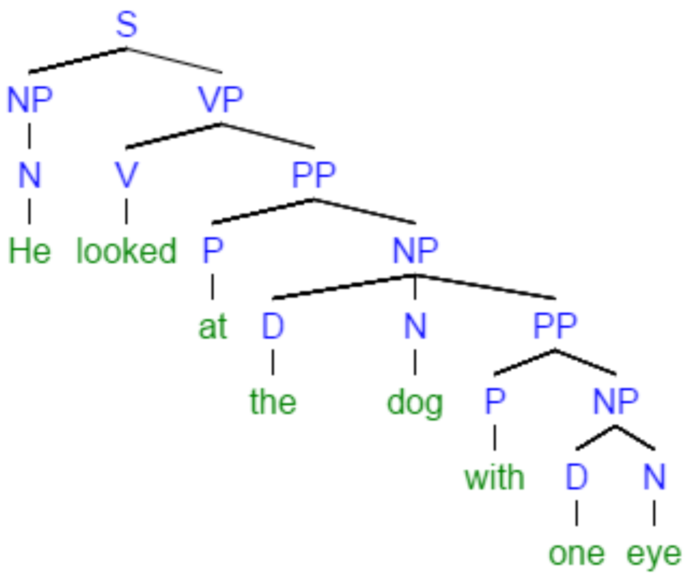
- Probabilistic selection:
  - $P(parse) = f(f_1(tree), f_2(t$
    - E.g. $P(parse) \propto exp(\sum_i \lambda_i f_i($
      - Examples of features

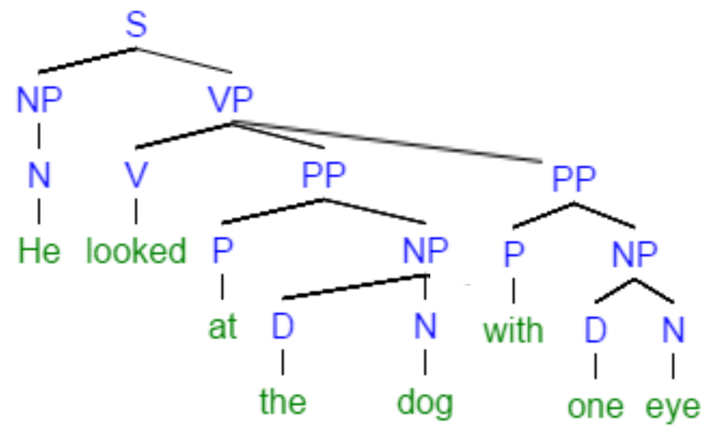
Disambiguating: Attempt 2



- $P(parse) = P(R_1, R_2, R_3, \ldots)$
- $P(parse) = P(R_1)P(R_2|R_1)P$
  - But this is a CFG.

Probabilistic Context Free Grammar





- Assign probability distributions ov
  - $VP \rightarrow V\ PP\ (0.2)$
  - $VP \rightarrow V\ PP\ PP\ (0.8)$

Probabilistic Context-Free Grammar

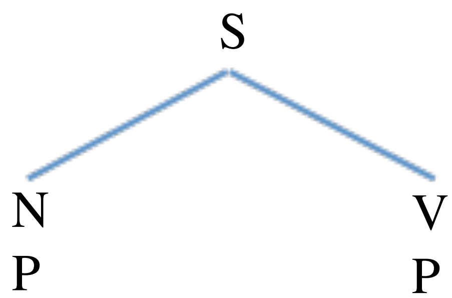
-
  - Associate a multinomial
    sides to the set of rules s
    - Conditional probability c
  - Generative story:
    1. Instantiate the start sy

Discrete time branching process

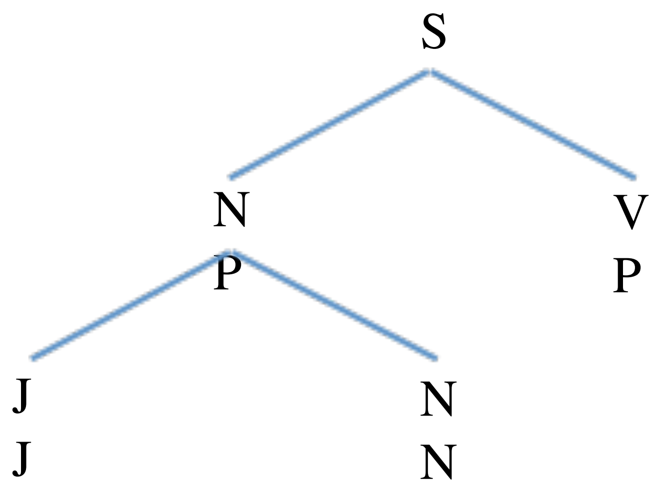
- Structure as the result of a **discrete time branching process**
  - Start in a known initial state, carry out stochastic steps (parameterized using multinomials) until some termination condition is met
  - Steps are (conditionally) independent of one another: probabilities multiply
  - *Total probability is the probability of the steps*

S

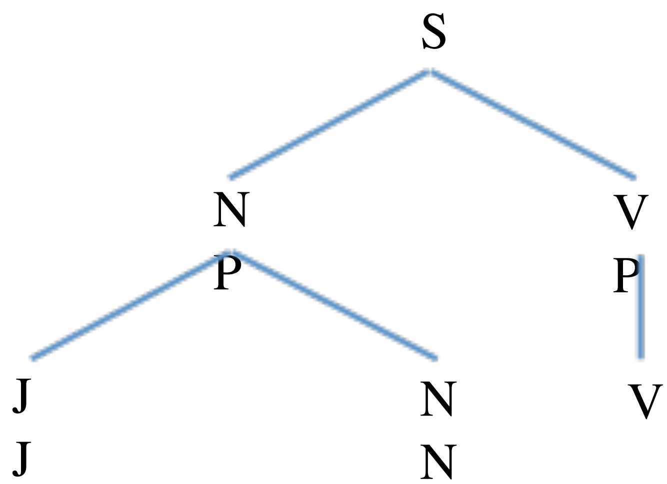
1.0



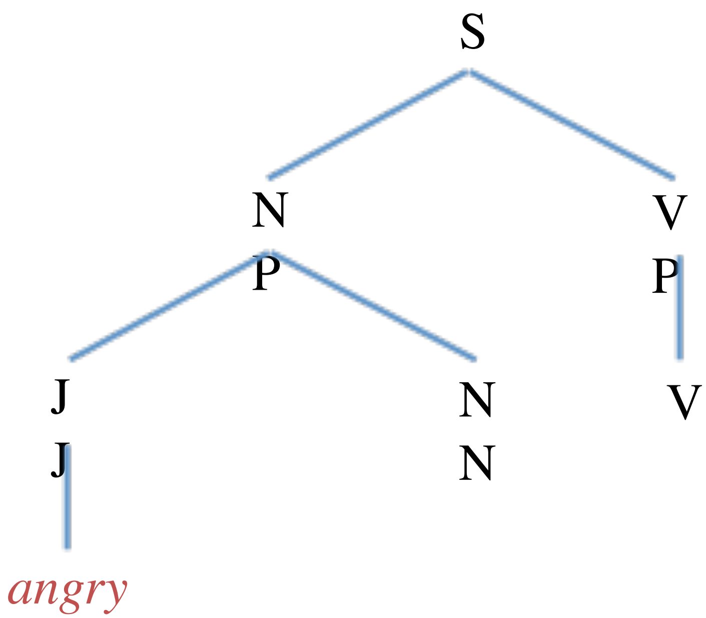
1.0 x p(NP VP | S)



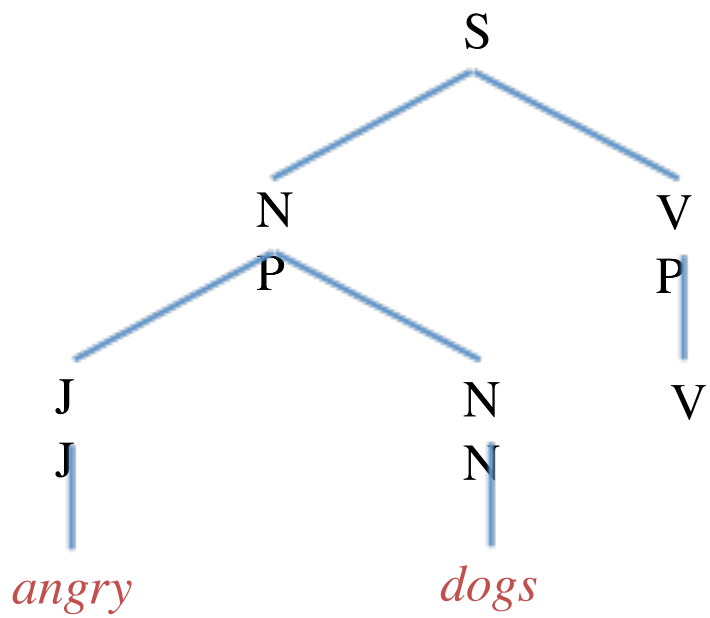
1.0 x p(NP VP | S)
x p(JJ NN | NP)



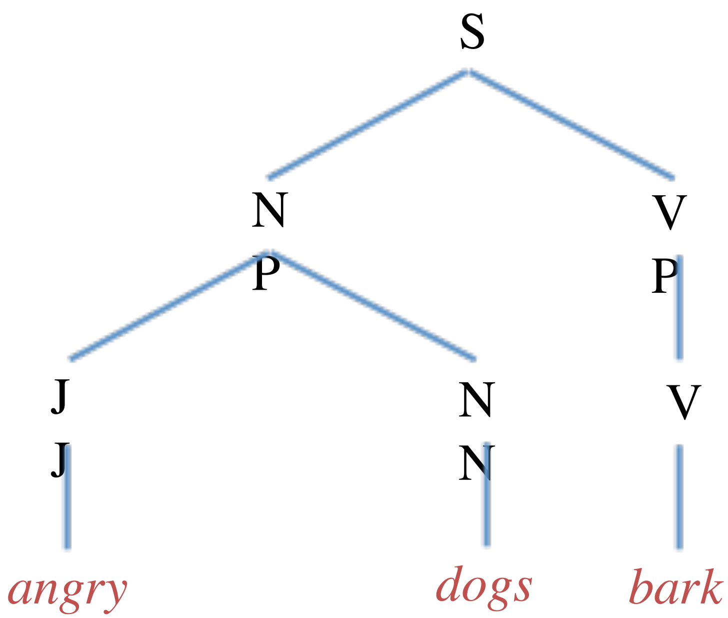
1.0 x p(NP VP | S)
x p(JJ NN | NP)
x p(V | VP)



1.0 x p(NP VP | S)
x p(JJ NN | NP)
x p(V | VP)
x p(*angry* | JJ)



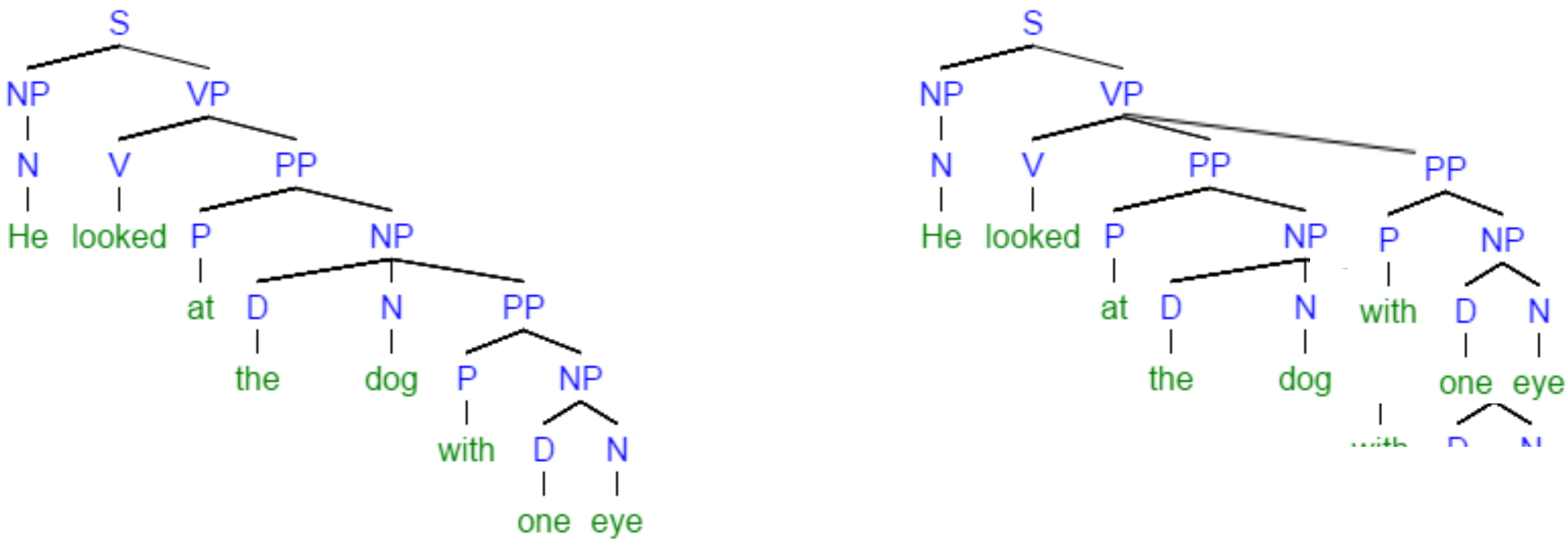
1.0 x p(NP VP | S)
x p(JJ NN | NP)
x p(V | VP)
x p(*angry* | JJ)
x p(*dogs* | NN)



1.0 x p(NP VP | S)
x p(JJ NN | NP)
x p(V | VP)
x p(*angry* | JJ)
x p(*dogs* | NN)
x p(*bark* | V)

$$p(\tau, \mathbf{x}) = \prod_{r \in \mathcal{G}} p(r \mid \mathcal{G})^{f(r \in \tau)}$$

Probabilistic Context Free Grammar



- Assign probability distributions ov
  - $VP \rightarrow V\ PP$ (0.2)
  - $VP \rightarrow V\ PP\ PP$ (0.8)

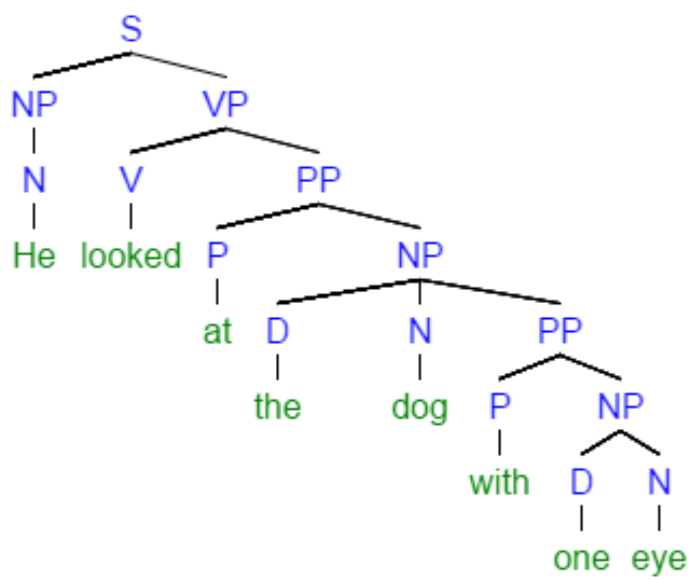
HMMs are Special PCFGs

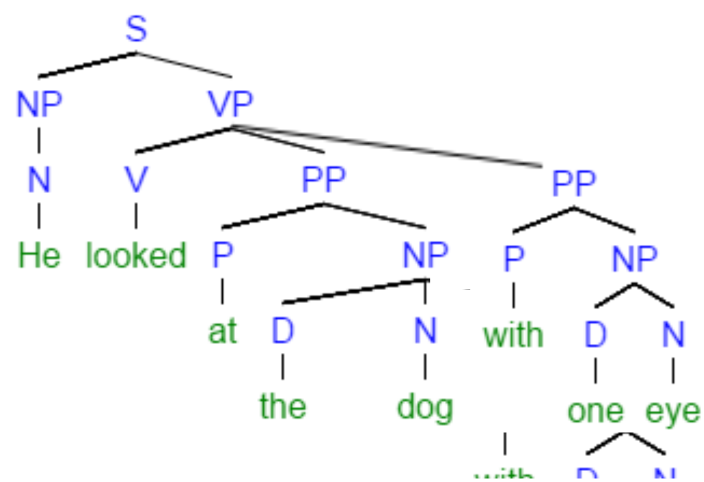
- (Actually HMMs are special PFSGs)
- Alphabet Σ
- N = HMM states Q
- Start state q0
- Rules

  $q \to x\ q'$ with probability pemit(x | q) ptrans(q' | q)

  $q \to \varepsilon$ with probability ptrans(stop | q)

Weighted Context Free Grammar





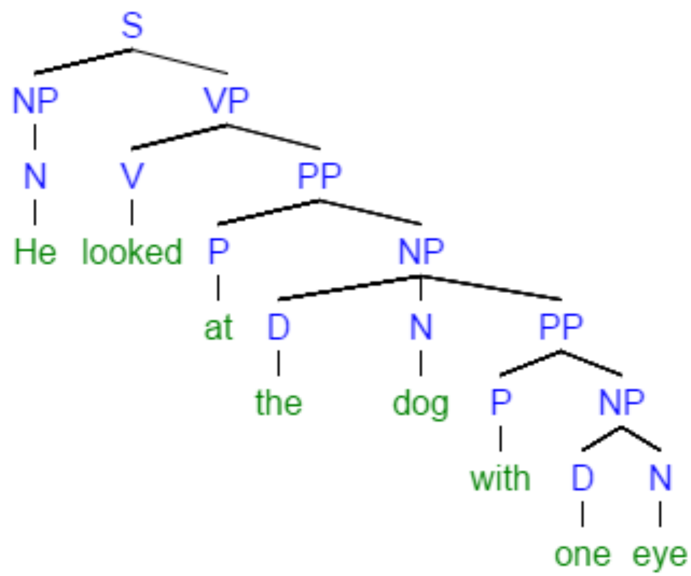
-
- Scores applied to rules n
  - Can just be weights
    - $VP \rightarrow V\ PP\ (-2)$

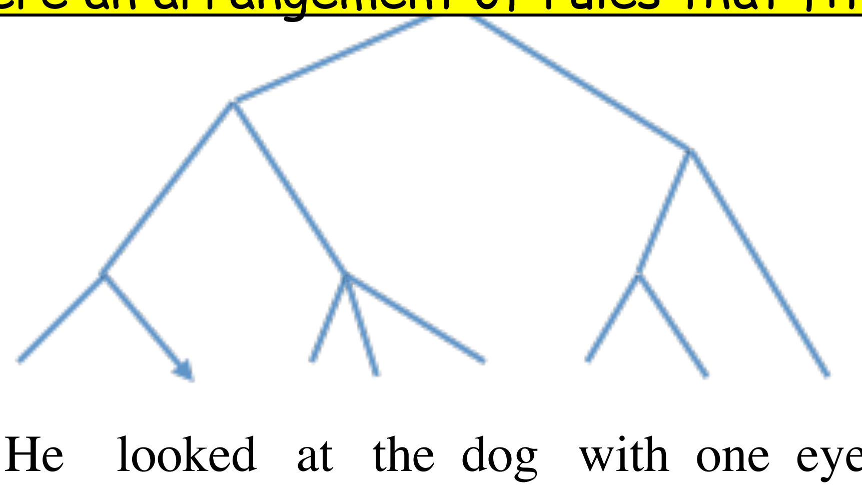
Weighted Context-Free Grammar

- Don't need a generative story; just assign weights to rules.

But where do the parse trees themselves come from?

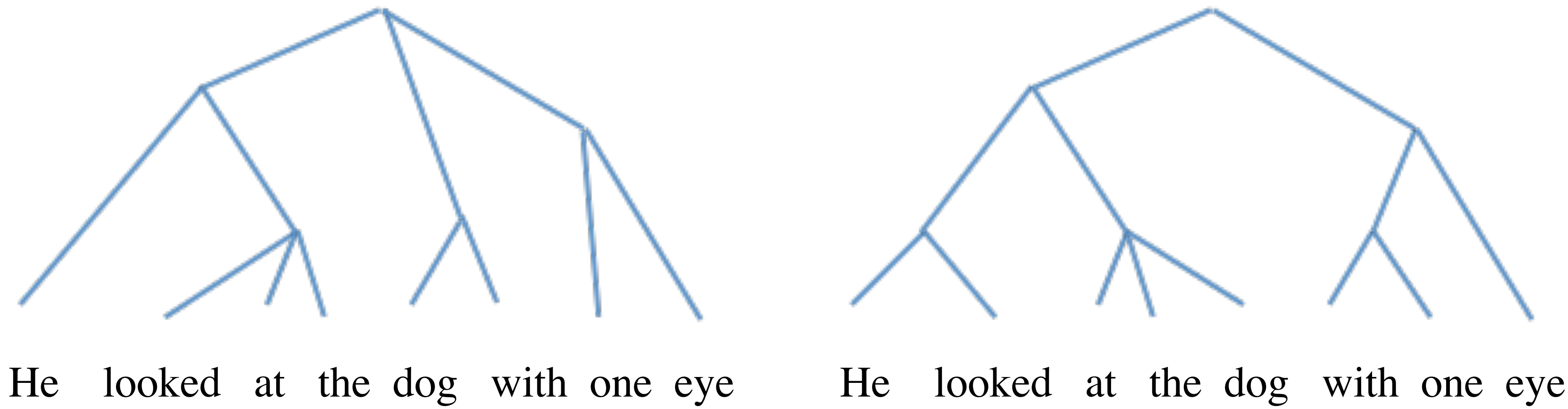
An arbitrarily drawn tree
Is there an arrangement of rules that fits this?





- How to hypothesize a parse tree for a sentence, given a CFG (or PCFG or WCFG)?
  - There are an exponentially large number

Parsing



- Consider every possible tree over the words
- Unambiguous grammar:
  - One of these trees aligns with the grammar
- Ambiguous grammar:
  - Find *a* tree that aligns with the grammar

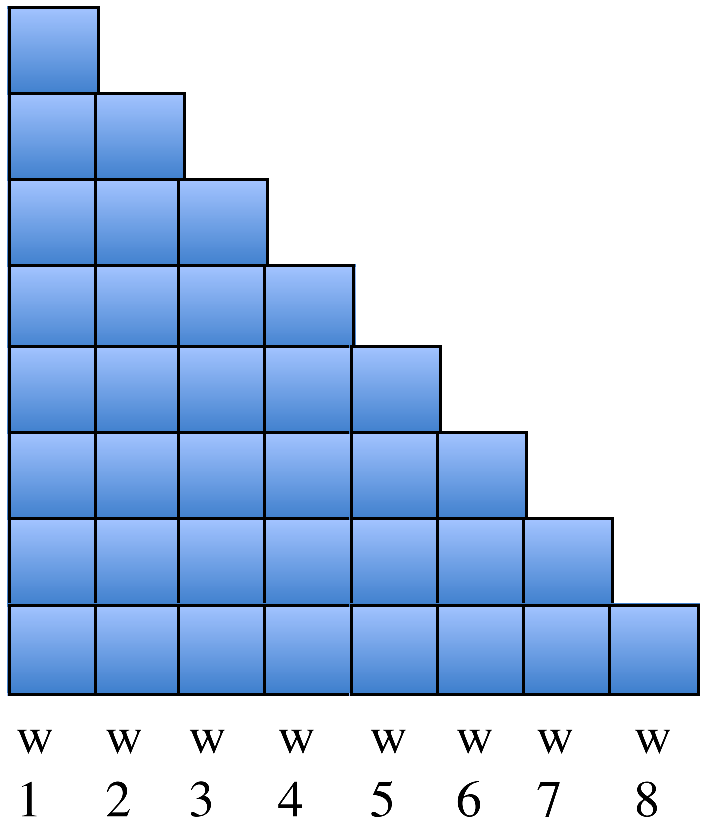
Some parsing algorithms

- CYK parser
  - (J. Cocke ‘70, D. Younger ‘67, T. Kasami ‘65)
- Earley’s parser

CYK parser: Unambiguous CFGs

- Explores every possible tree, but does so with a dynamic program
- To keep computation down, works only with CNF grammars
  - Recall that every CFG can be rewritten as a CNF
  - Result of CNF formalism: Every node a tree *must* connect with either a node to the immediate left or the immediate right
  - Result of contiguity constraint in grammar: Every node *must* represent the entire sequence of words below it

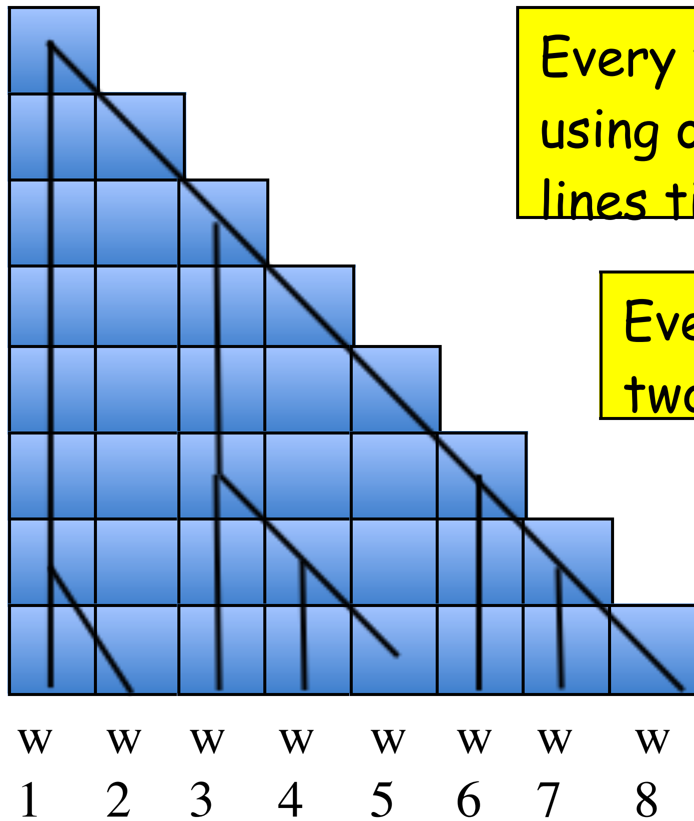
CYK : Unambiguous CFGs



- Given a grammar and a word sequence w1…wN:
- Construct this triangle (number of rows = number of words)

CYK : Unambiguous CFGs

Every possible tree can be represented within this grid



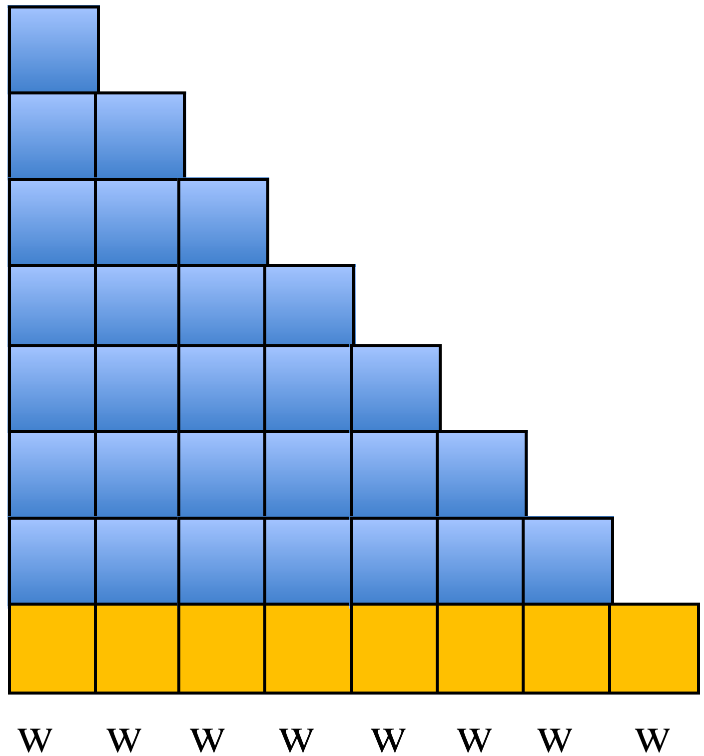
Every tree can be drawn using only vertical lines and lines tilted left 45o

Every "node" merges exactly two lines (CNF grammar)

Every "node" spans all words in the triangle below it

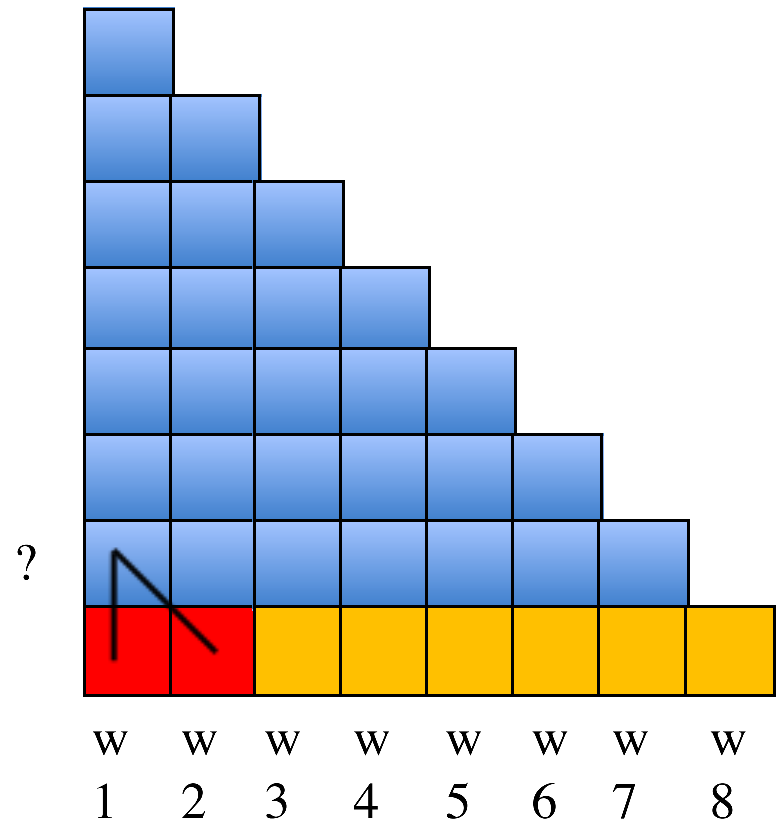
- Given a grammar and a word sequence w1…wN:
- Construct this triangle (number of rows =

CYK : Unambiguous CFGs



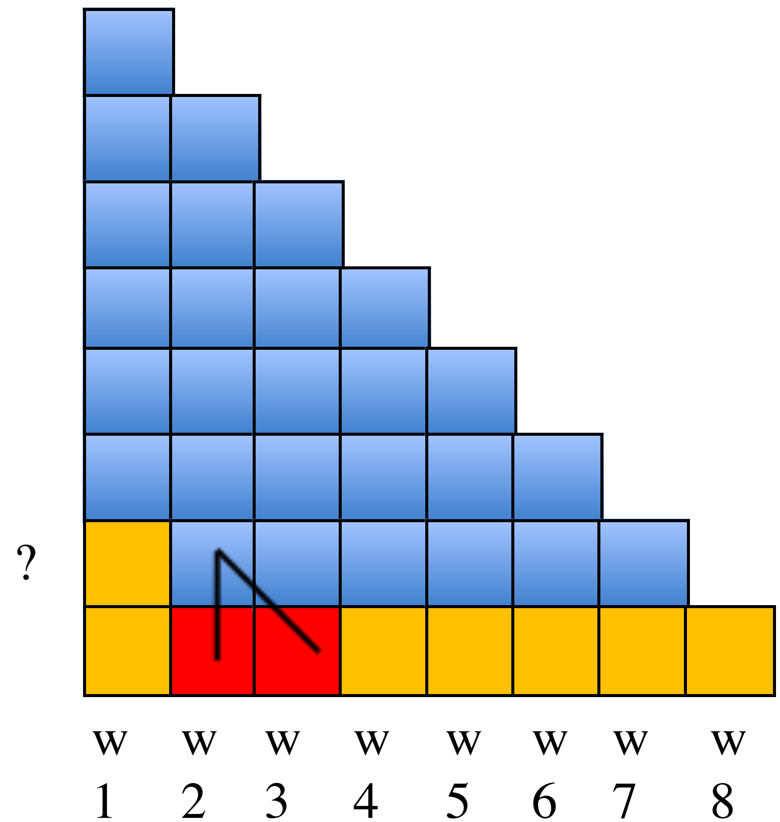
- For each word in the first row, find all production rules that can produce it.
  - Store (pointers to) all in the corresponding block

CYK : Unambiguous CFGs



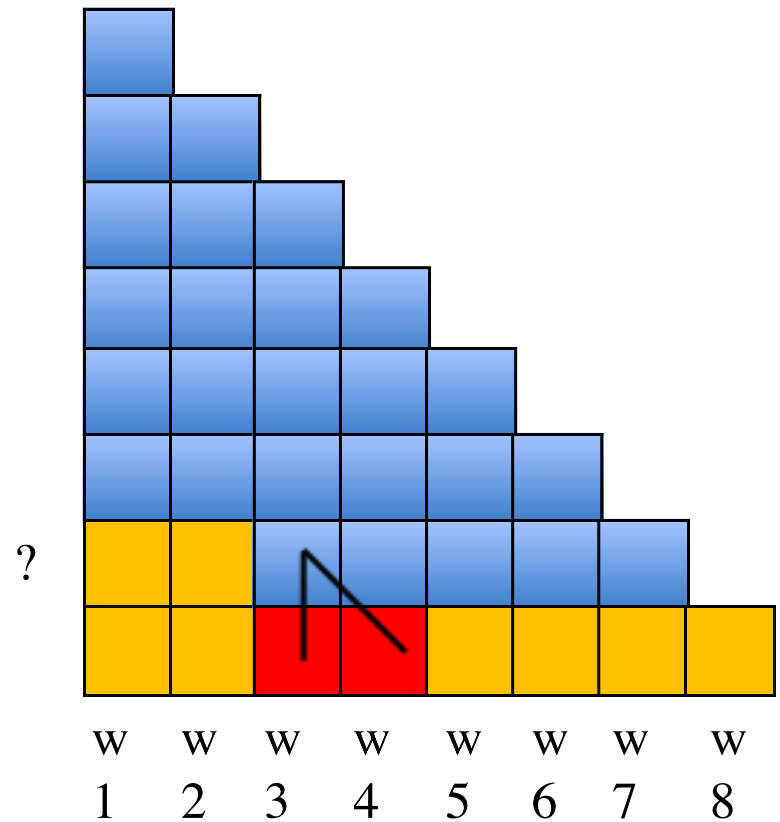
- For each block in the next row, find all rules that produce any combination of the non-terminals immediately below it

CYK : Unambiguous CFGs



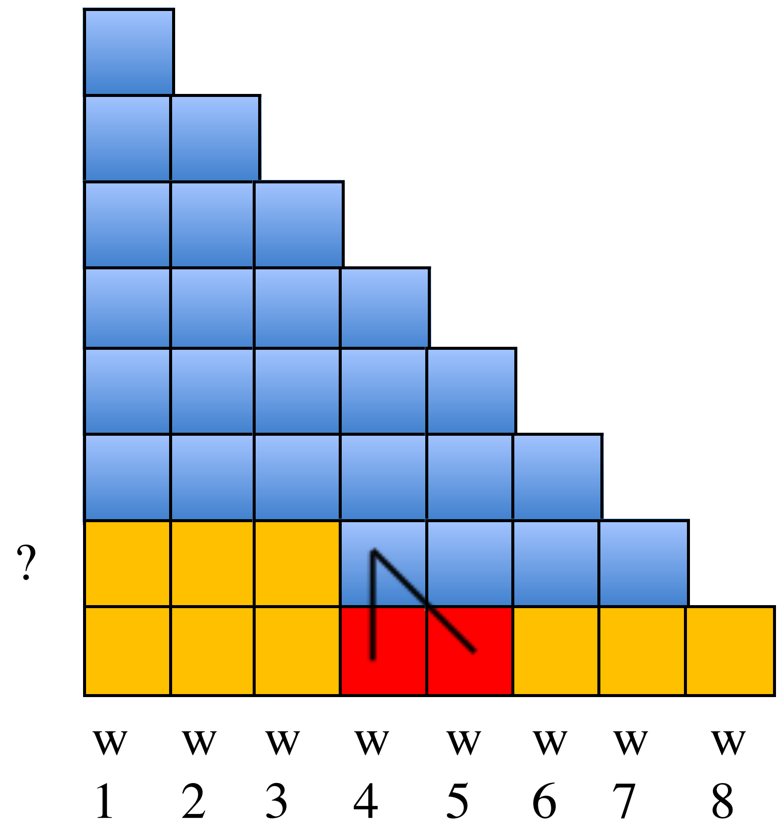
- For each block in the next row, find all rules that produce any combination of the non-terminals immediately below it

CYK : Unambiguous CFGs



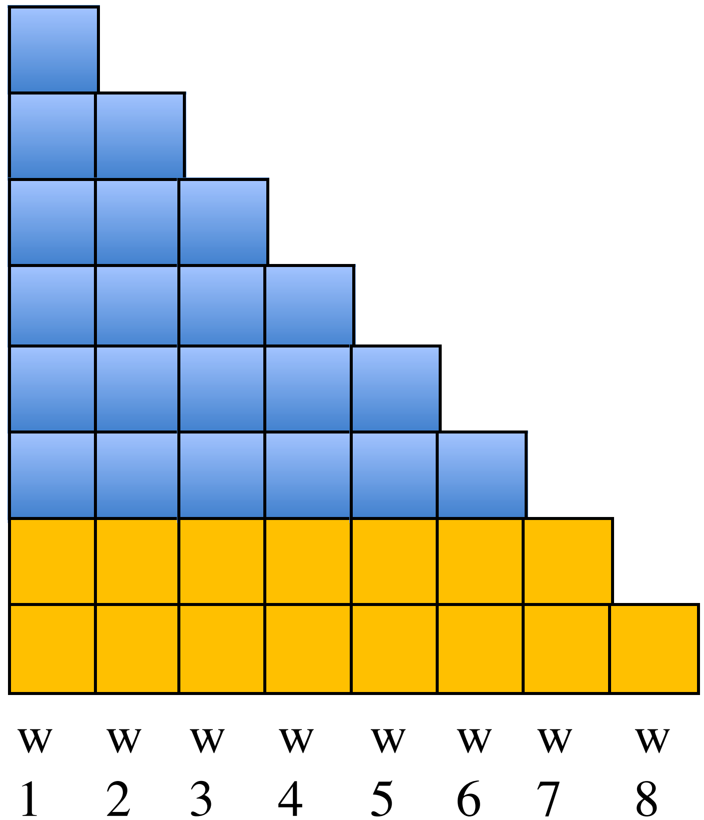
- For each block in the next row, find all rules that produce any combination of the non-terminals immediately below it

CYK : Unambiguous CFGs



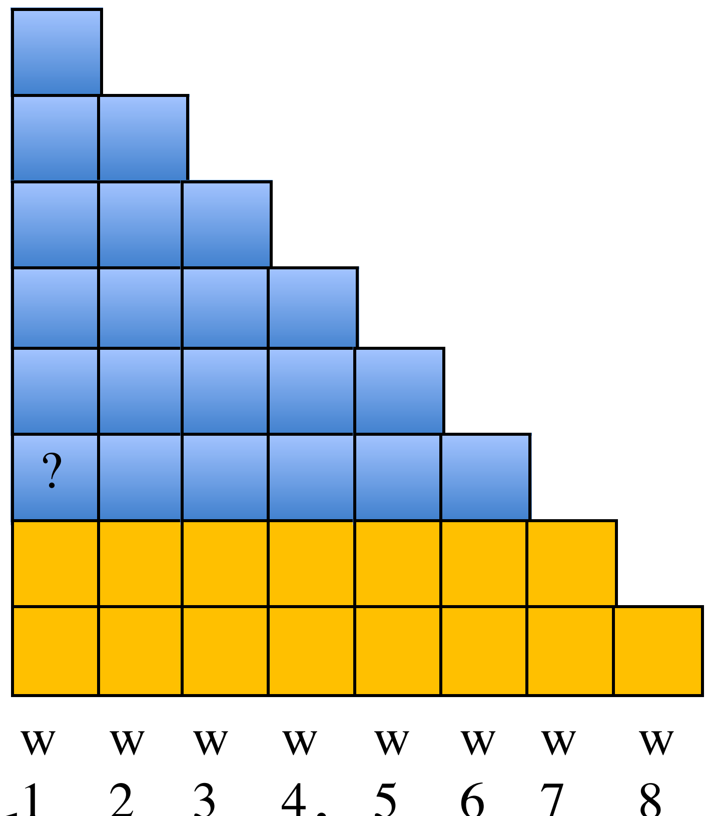
- For each block in the next row, find all rules that produce any combination of the non-terminals immediately below it

CYK : Unambiguous CFGs



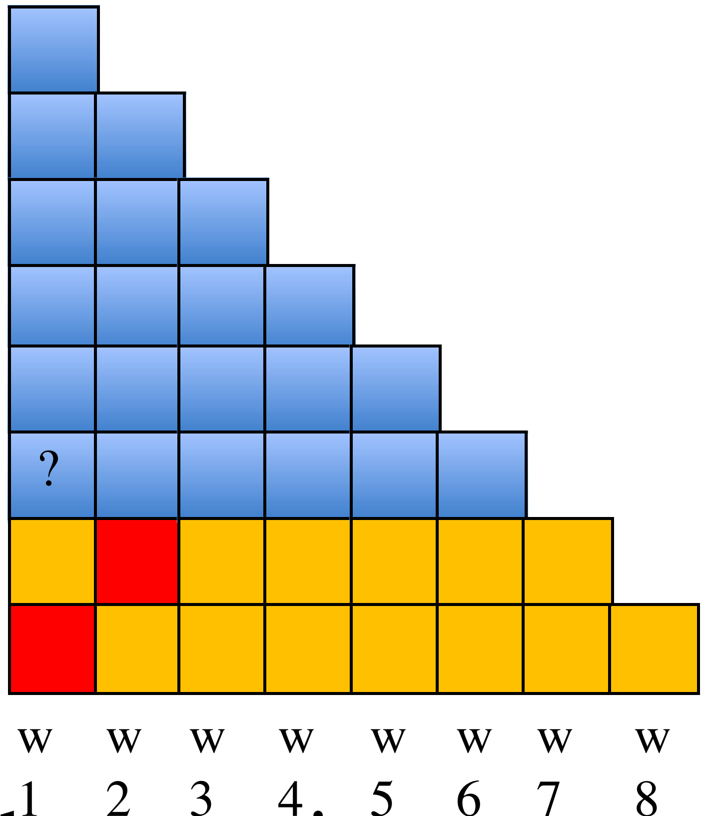
- For each block in the next row, find all rules that produce any combination of the non-terminals immediately below it

CYK : Unambiguous CFGs



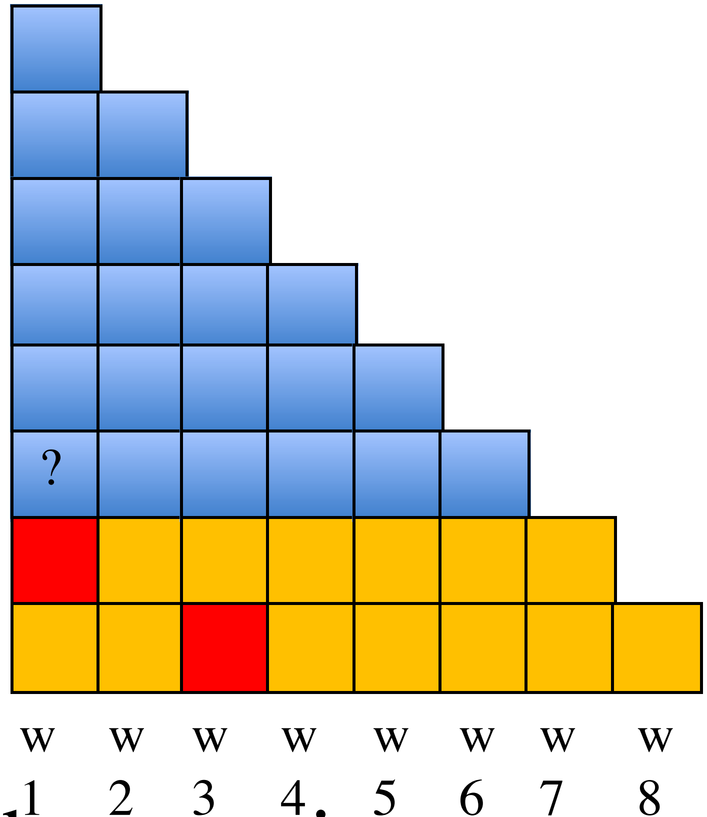
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



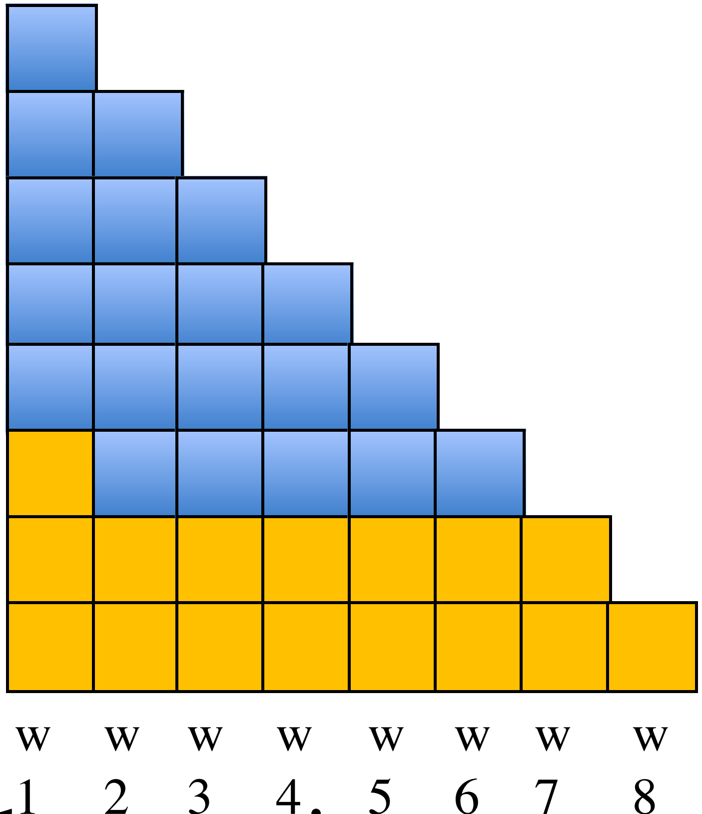
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



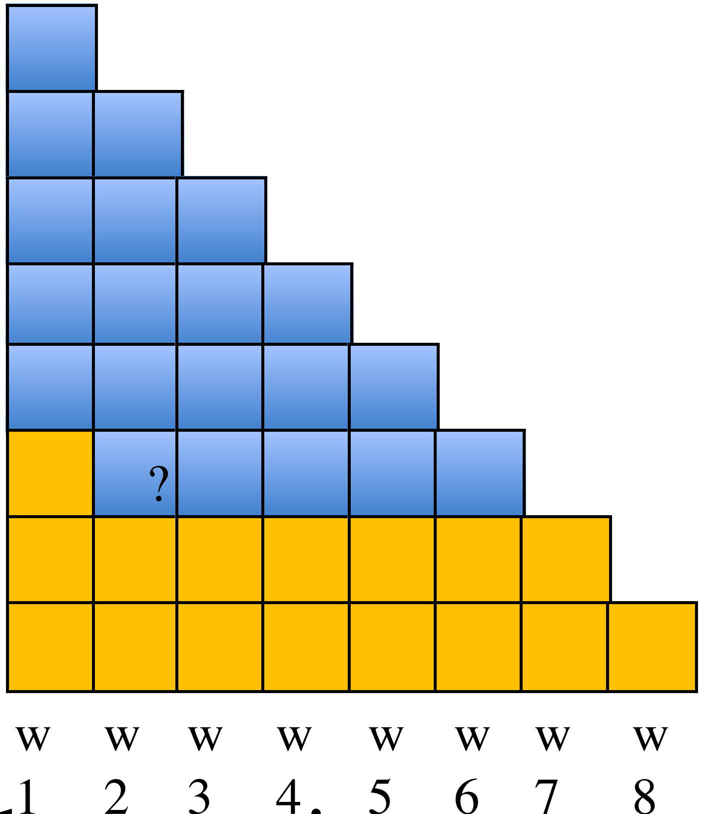
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



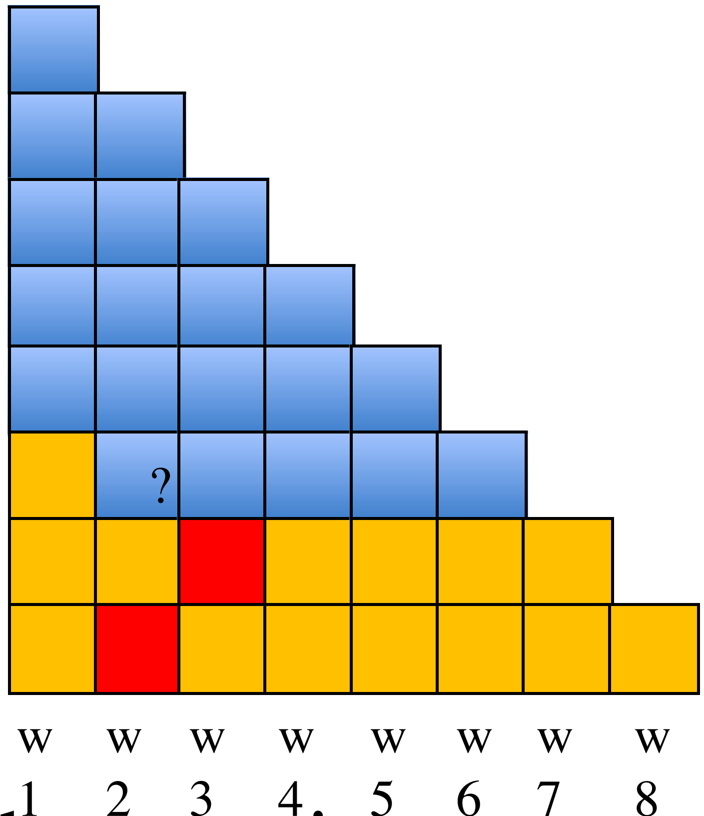
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



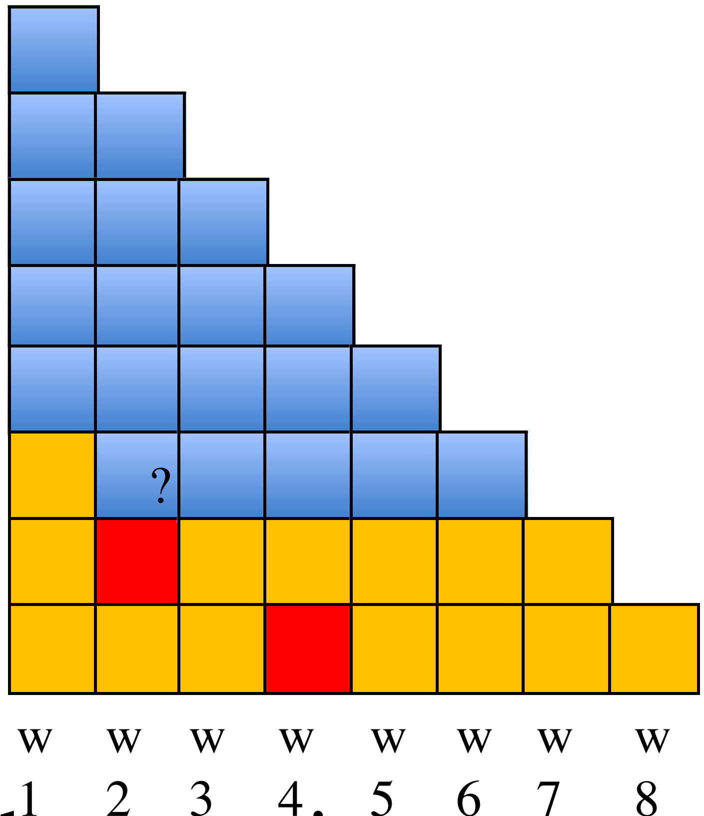
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



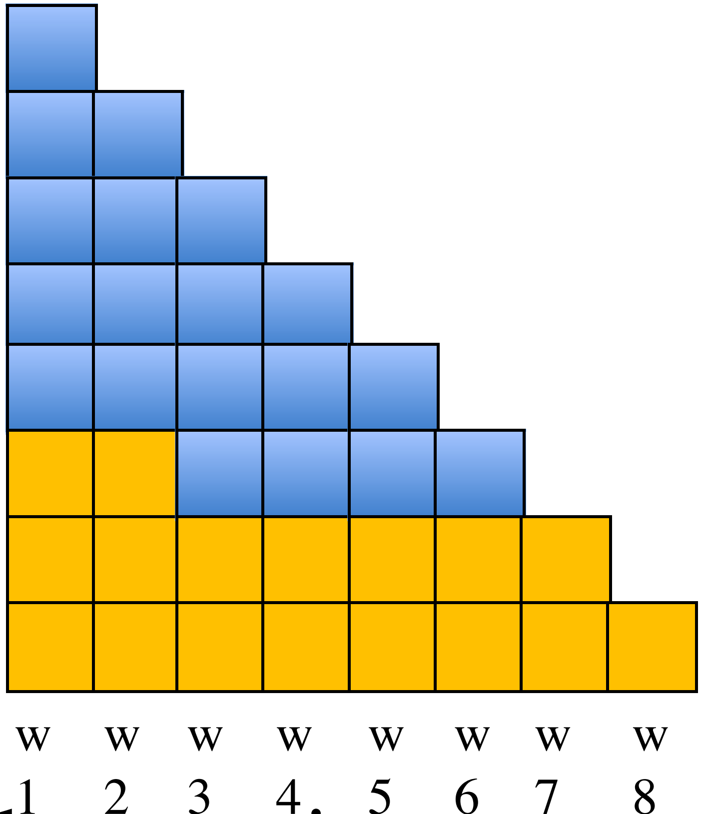
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



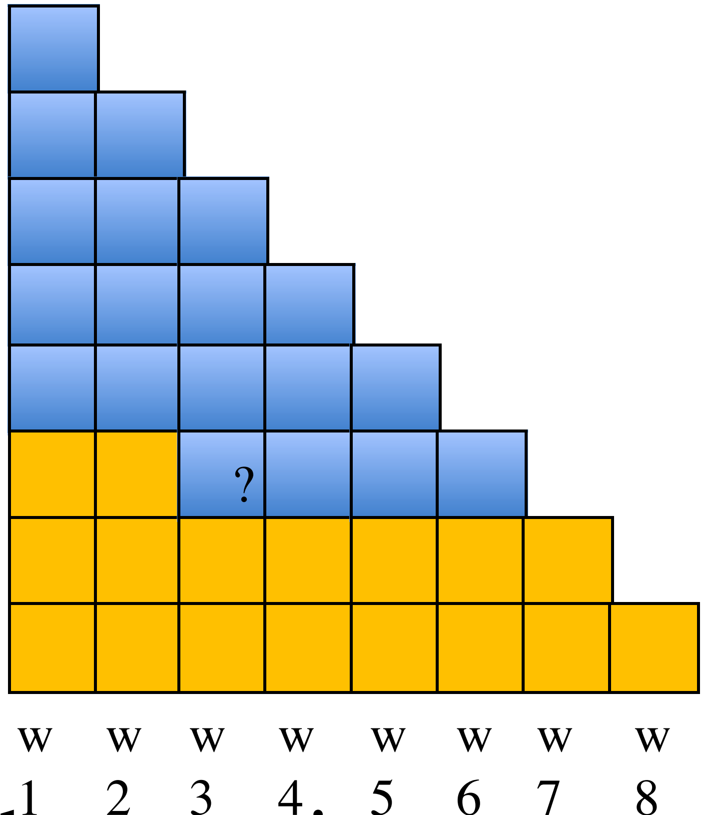
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



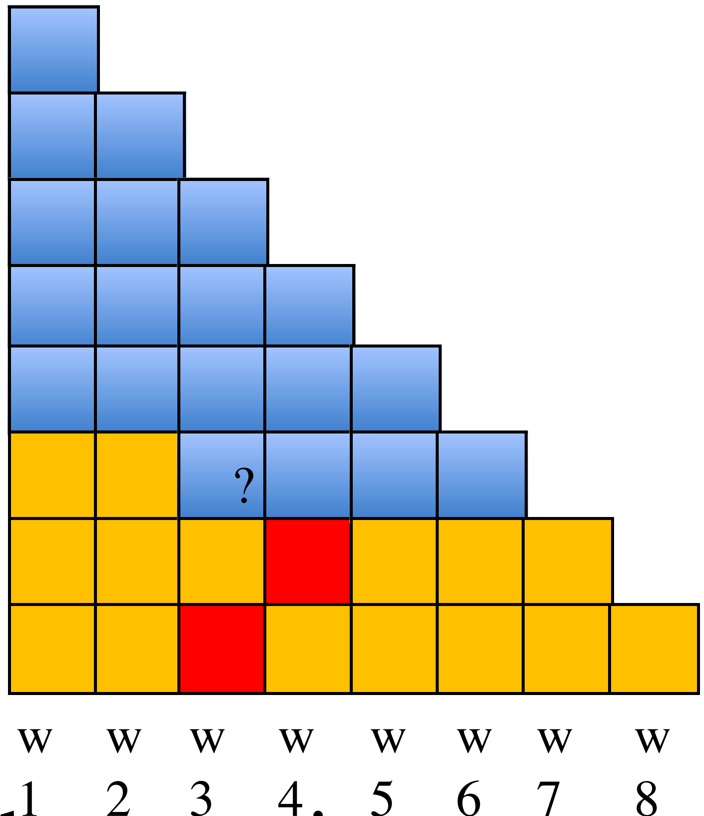
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



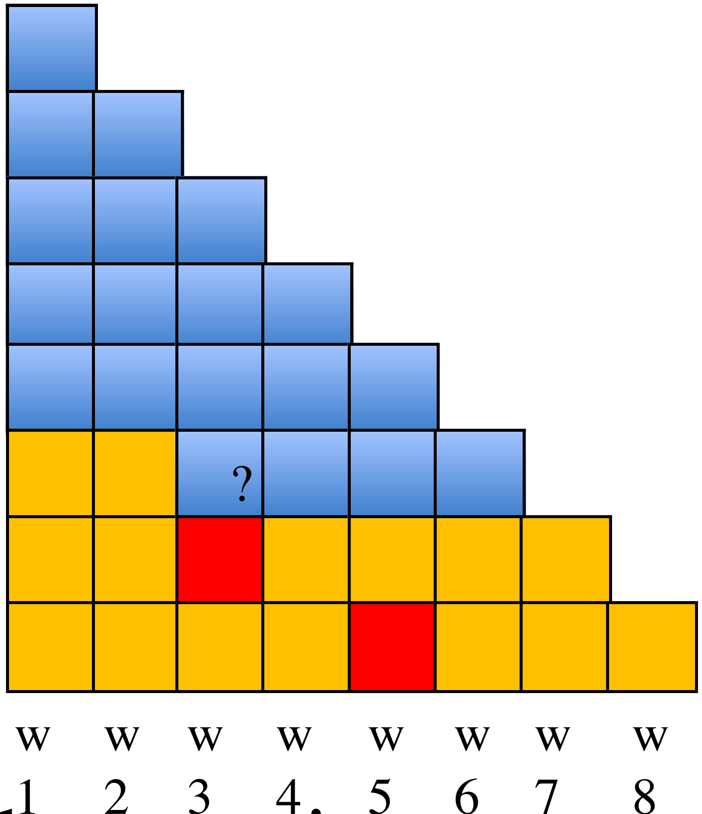
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



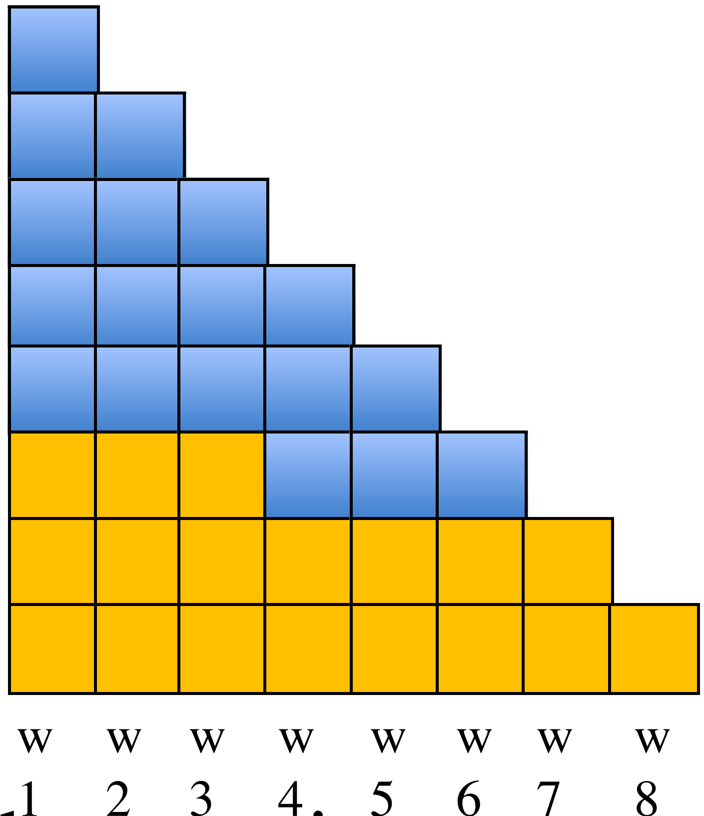
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



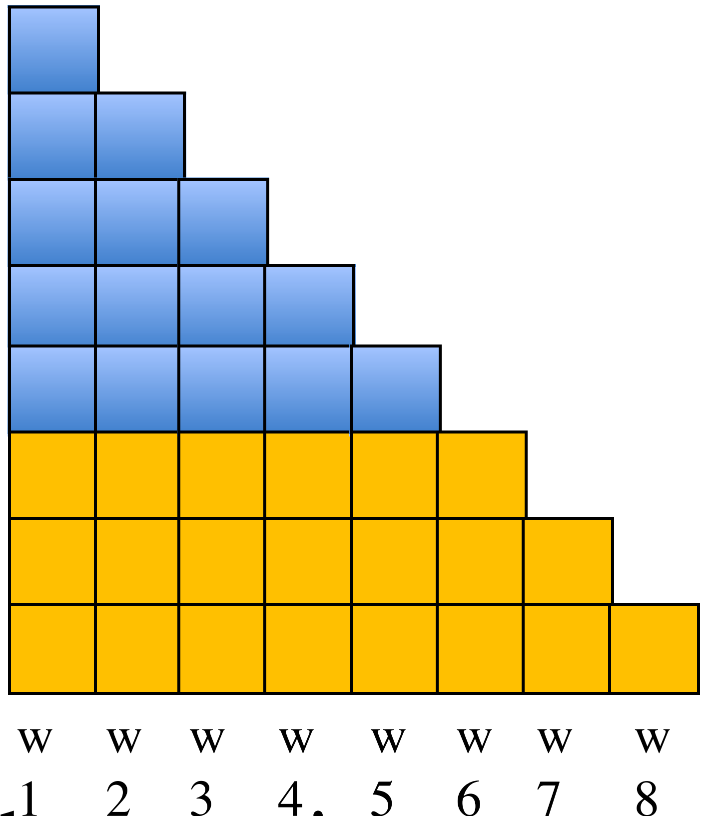
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



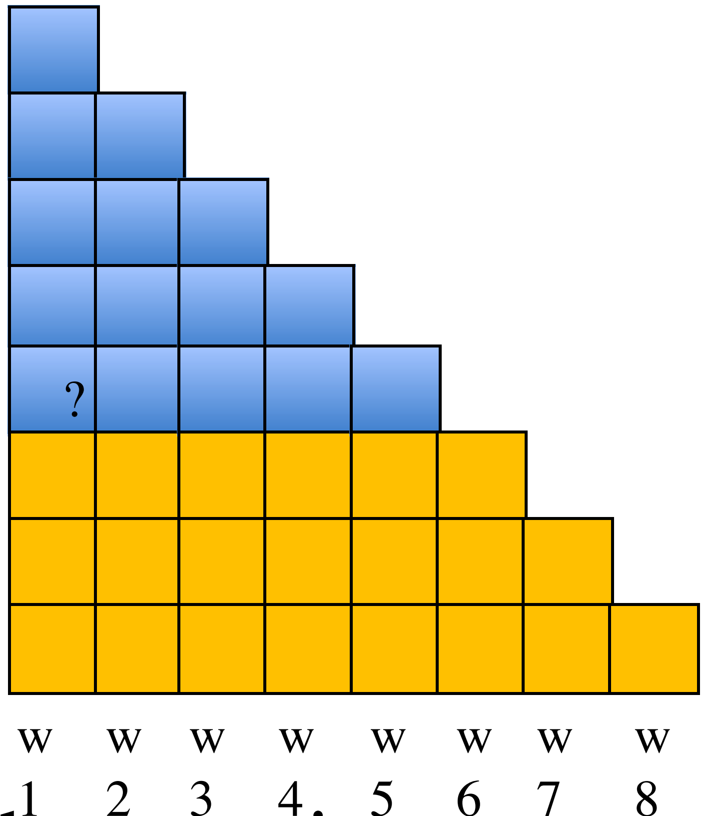
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



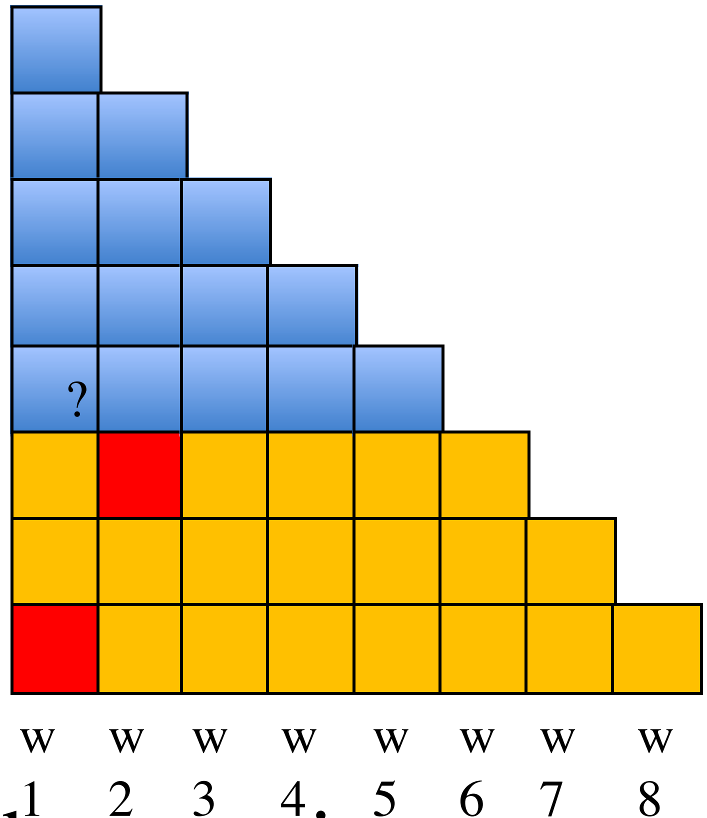
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



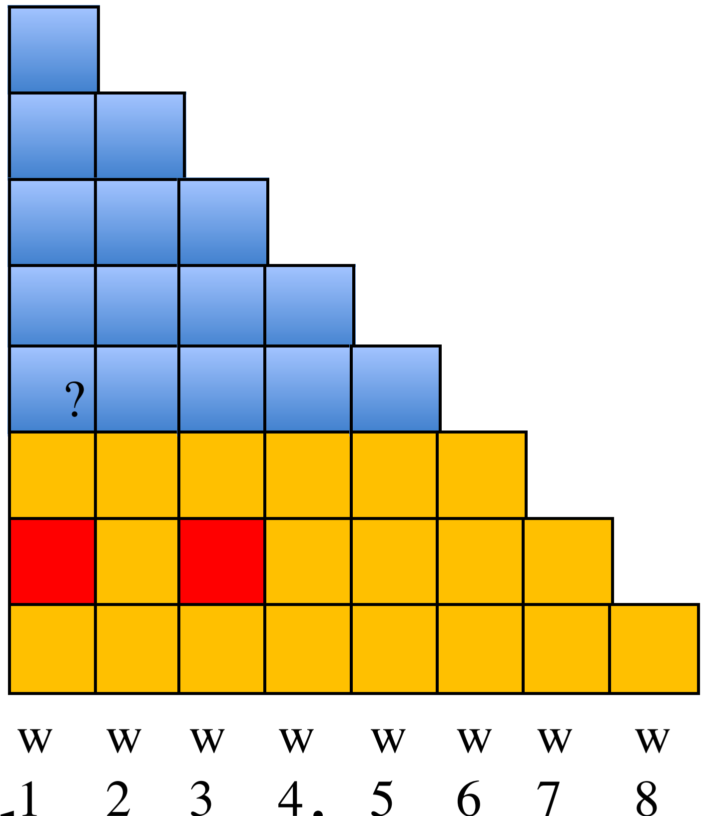
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



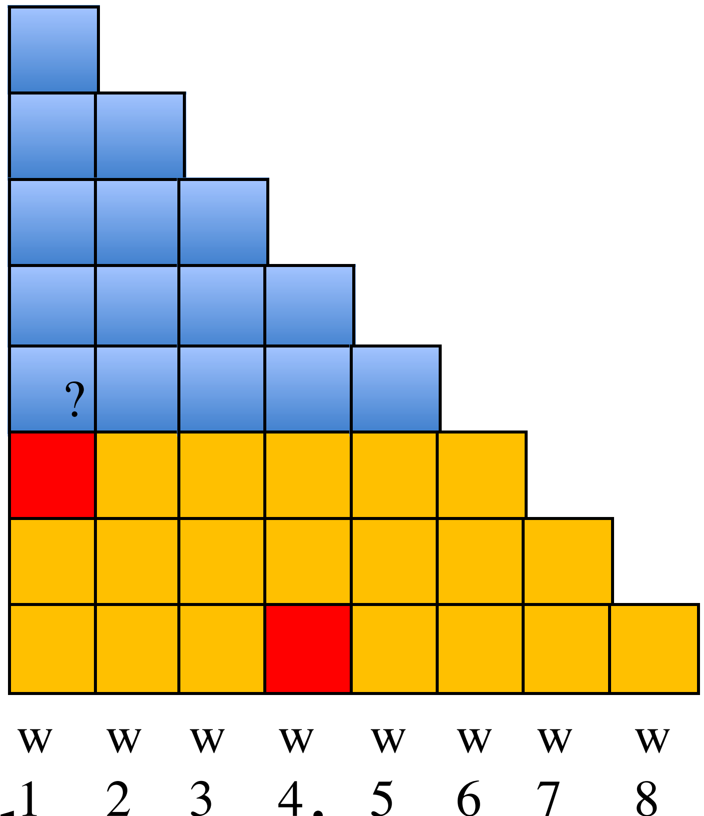
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



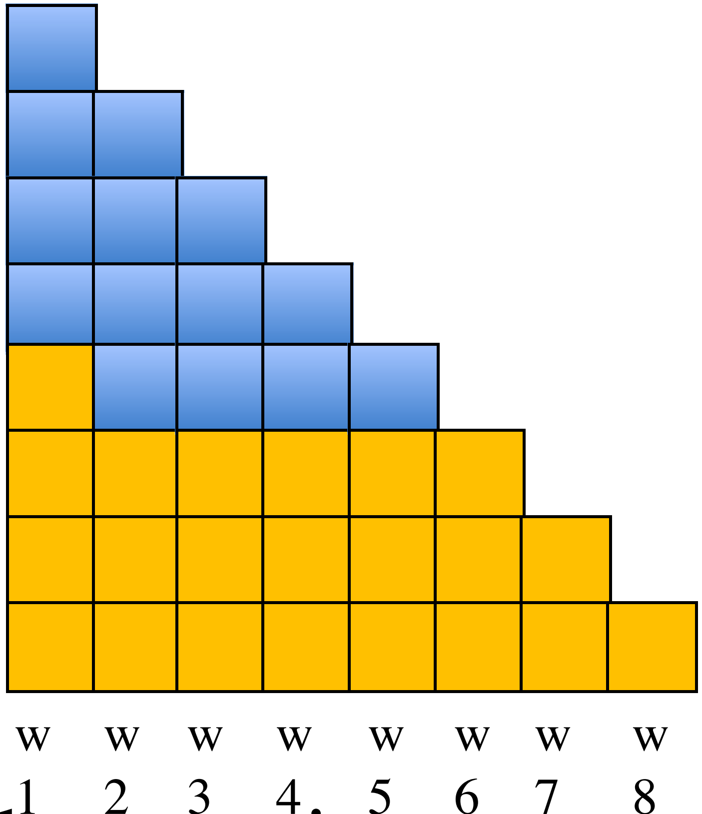
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



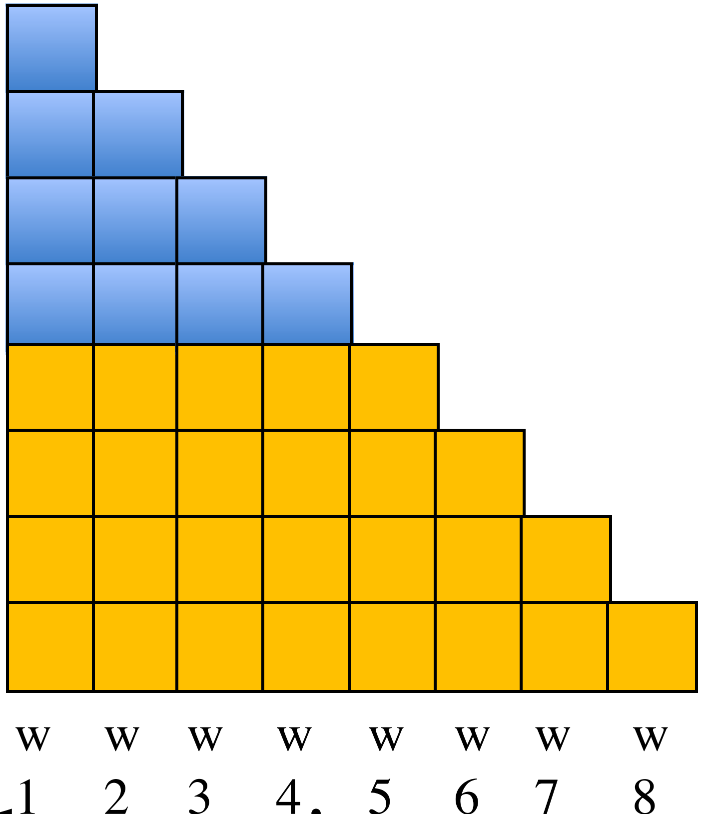
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



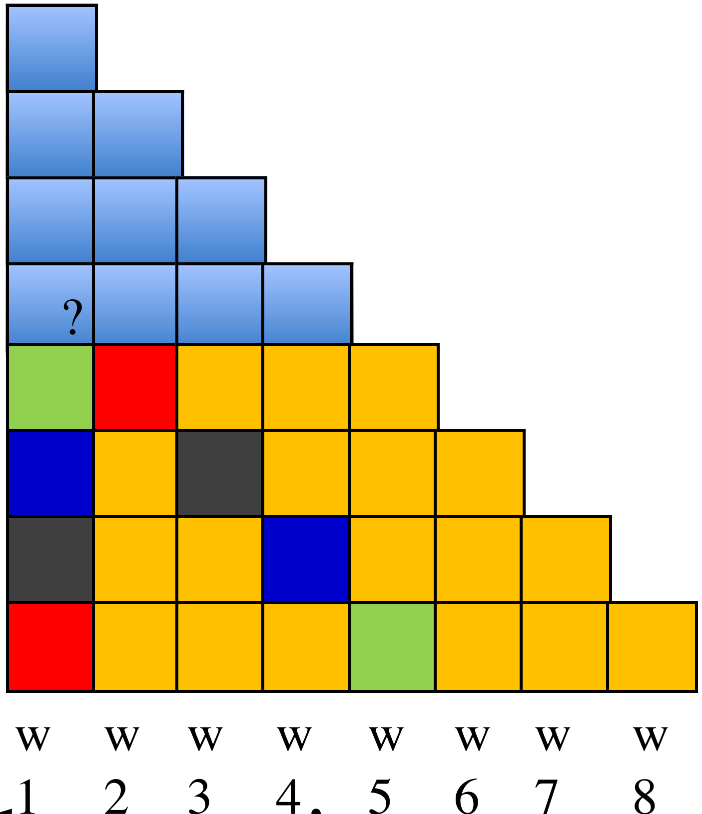
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



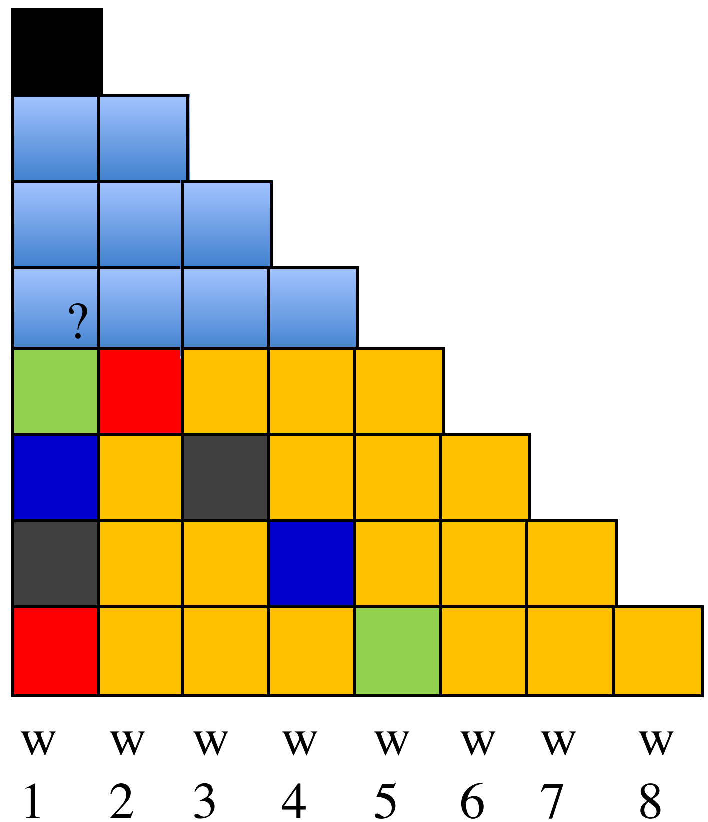
- For each higher row in sequence
  - For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - Identify any rules that produce any combination of NTs for the

CYK : Unambiguous CFGs



- For each higher row in sequence
  - – For each block
    - For each pair of lower nodes that can span the entire section of words represented by the block
      - – Identify any rules that produce any combination of NTs for the

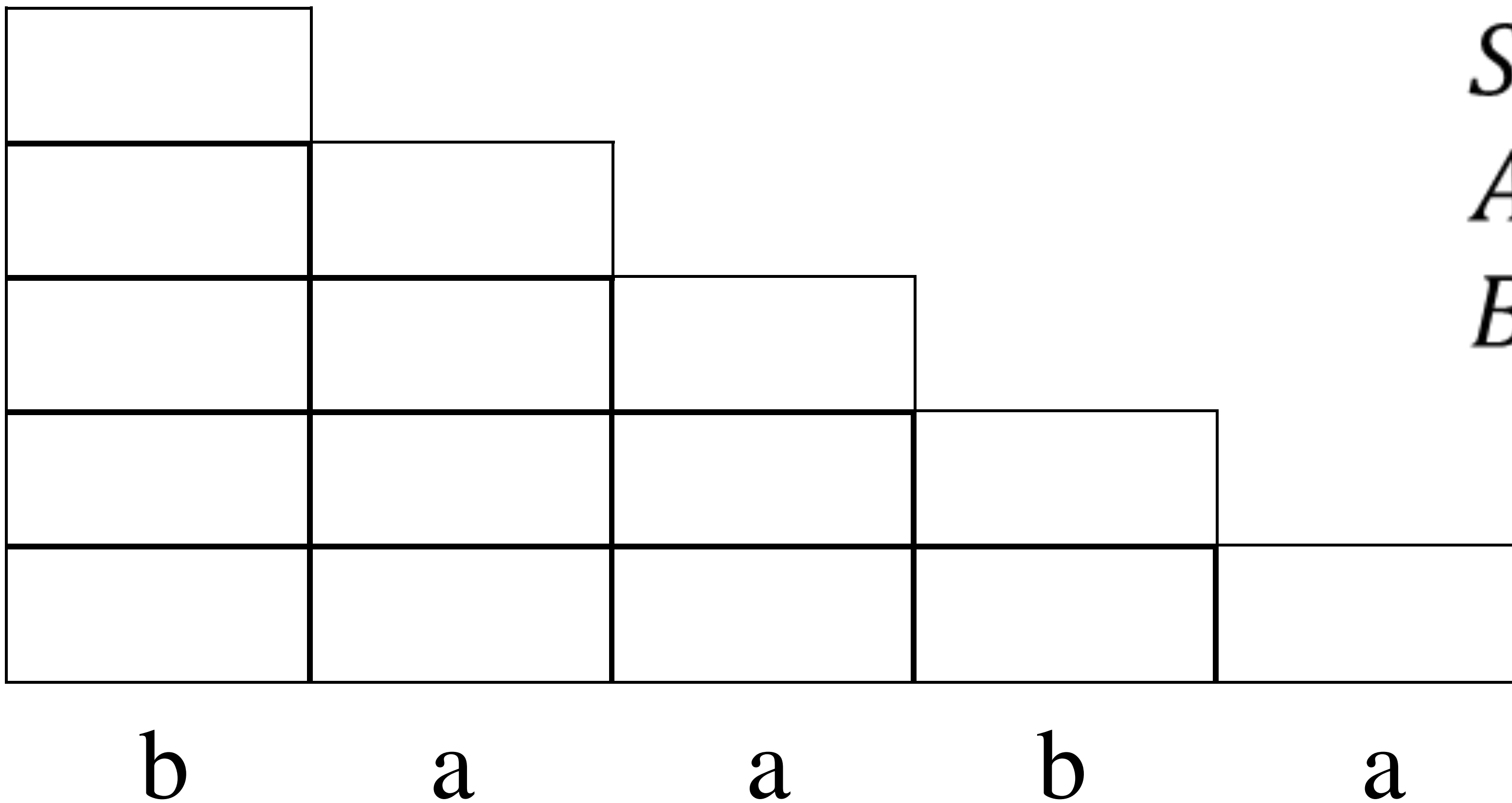
CYK : Unambiguous CFGs



- If, eventually, the top box is populated, the string belongs to the language

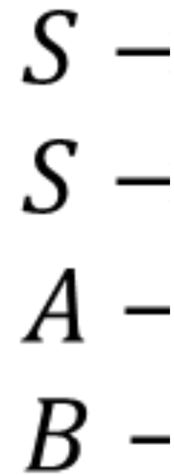
CYK example

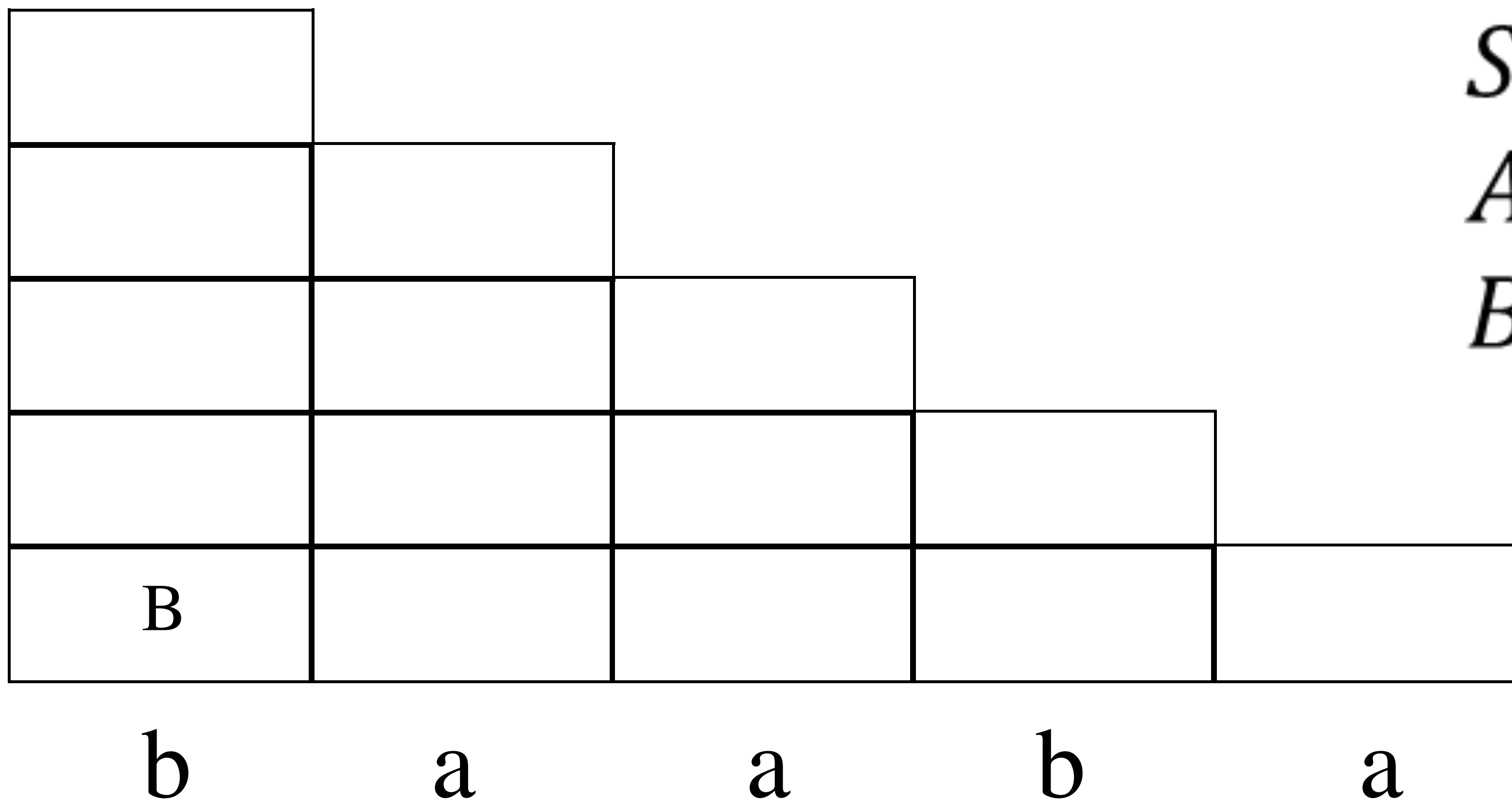
$S -$
$S -$
$A -$
$B -$



- From slides I found on a UC Davis website

CYK example





- From slides I found on a UC Davis website

CYK example

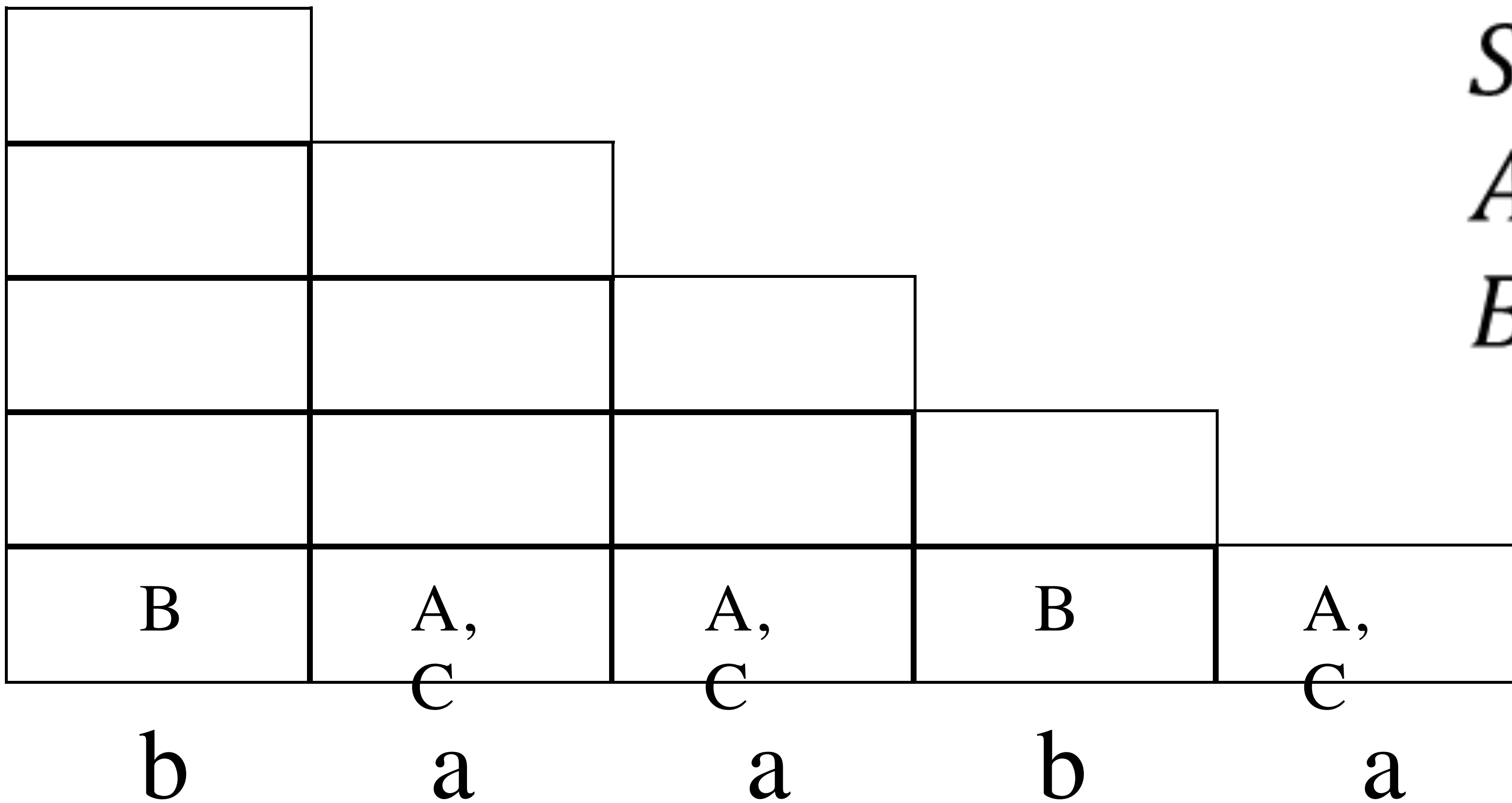
$S -$
$S -$
$A -$
$B -$

B	A, C			
b	a	a	b	a

- From slides I found on the UC Davis website

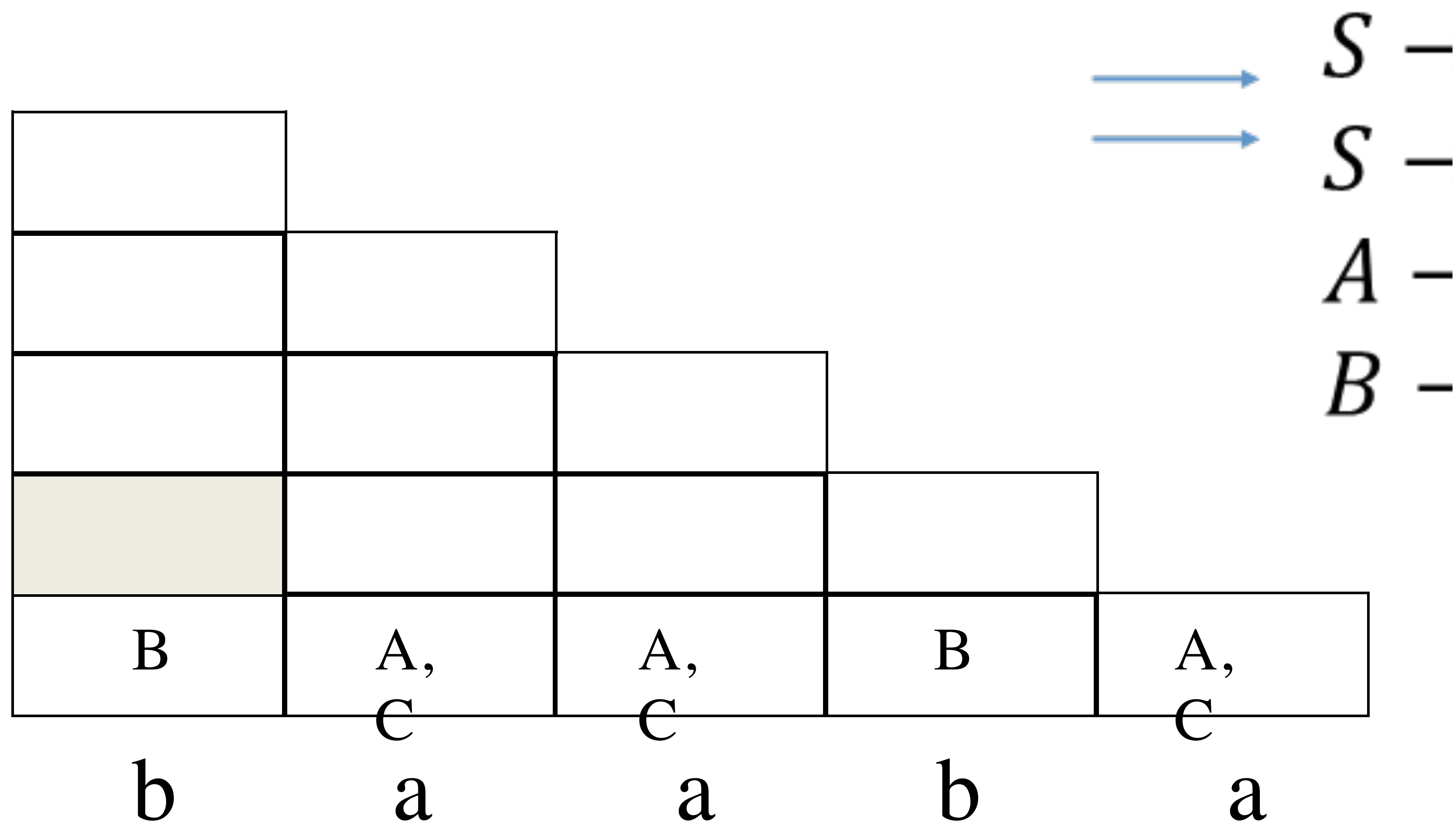
CYK example

$S -$
$S -$
$A -$
$B -$



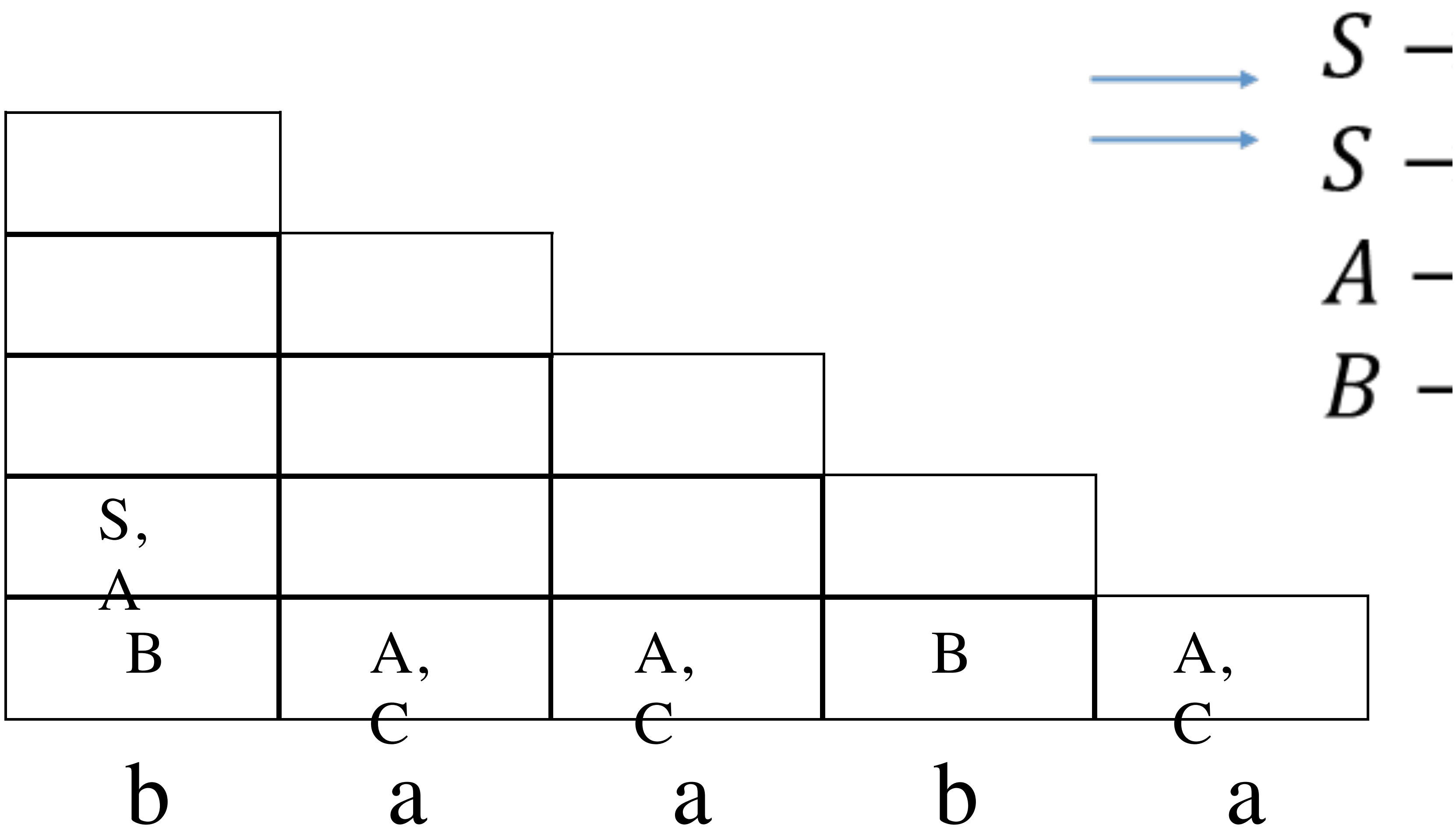
- From slides I found on the UC Davis website

CYK example



- Possible productions: B A, B C
- We find two rules for this

CYK example



- Possible productions: B A, B C
- We find two rules for this

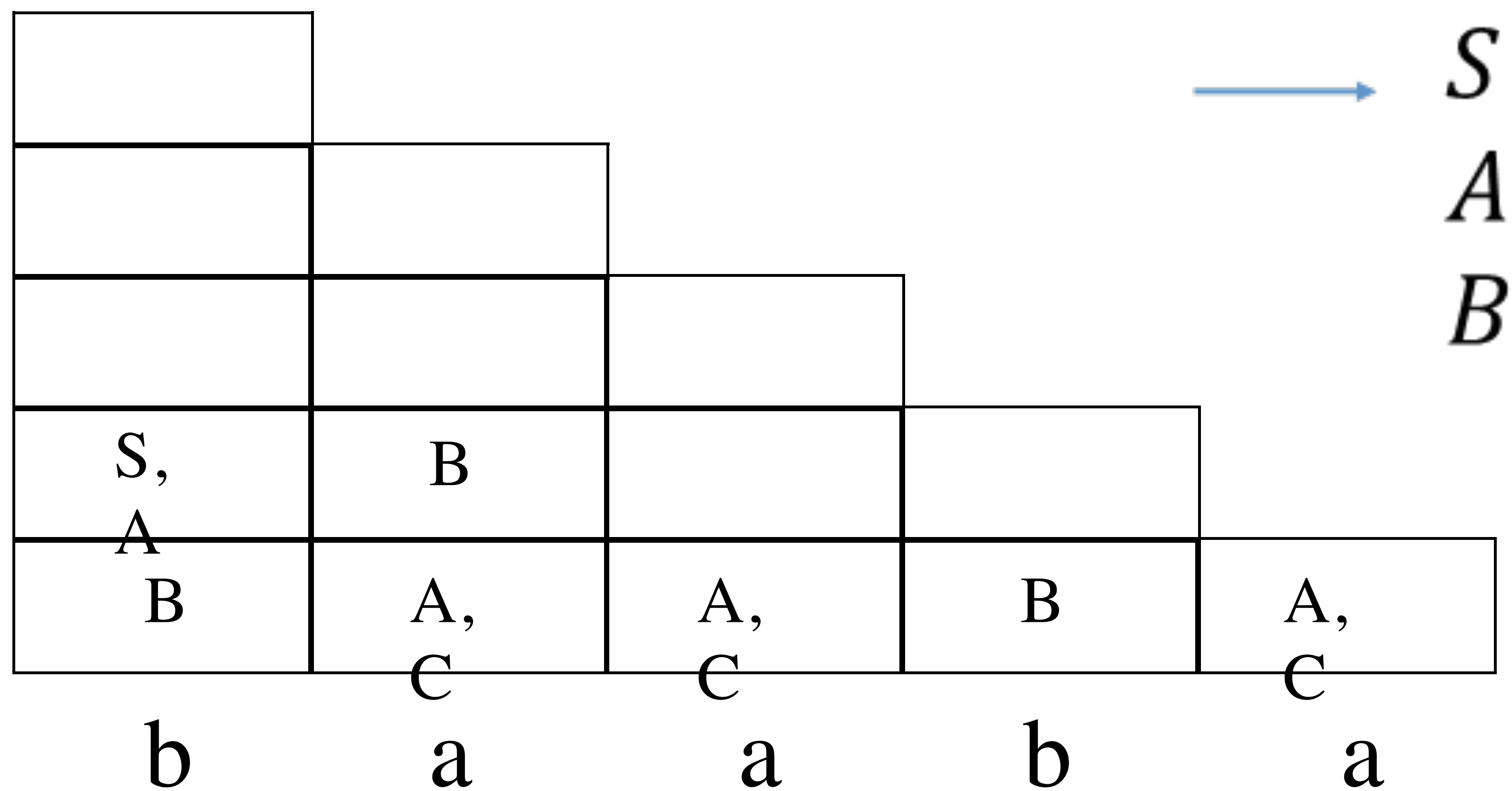
CYK example

S –
S –
A –
B –

S, A				
B	A, C	A, C	B	A, C
b	a	a	b	a

- Possible productions: A A, A C, C A, C C
- One rule

CYK example



$S -$

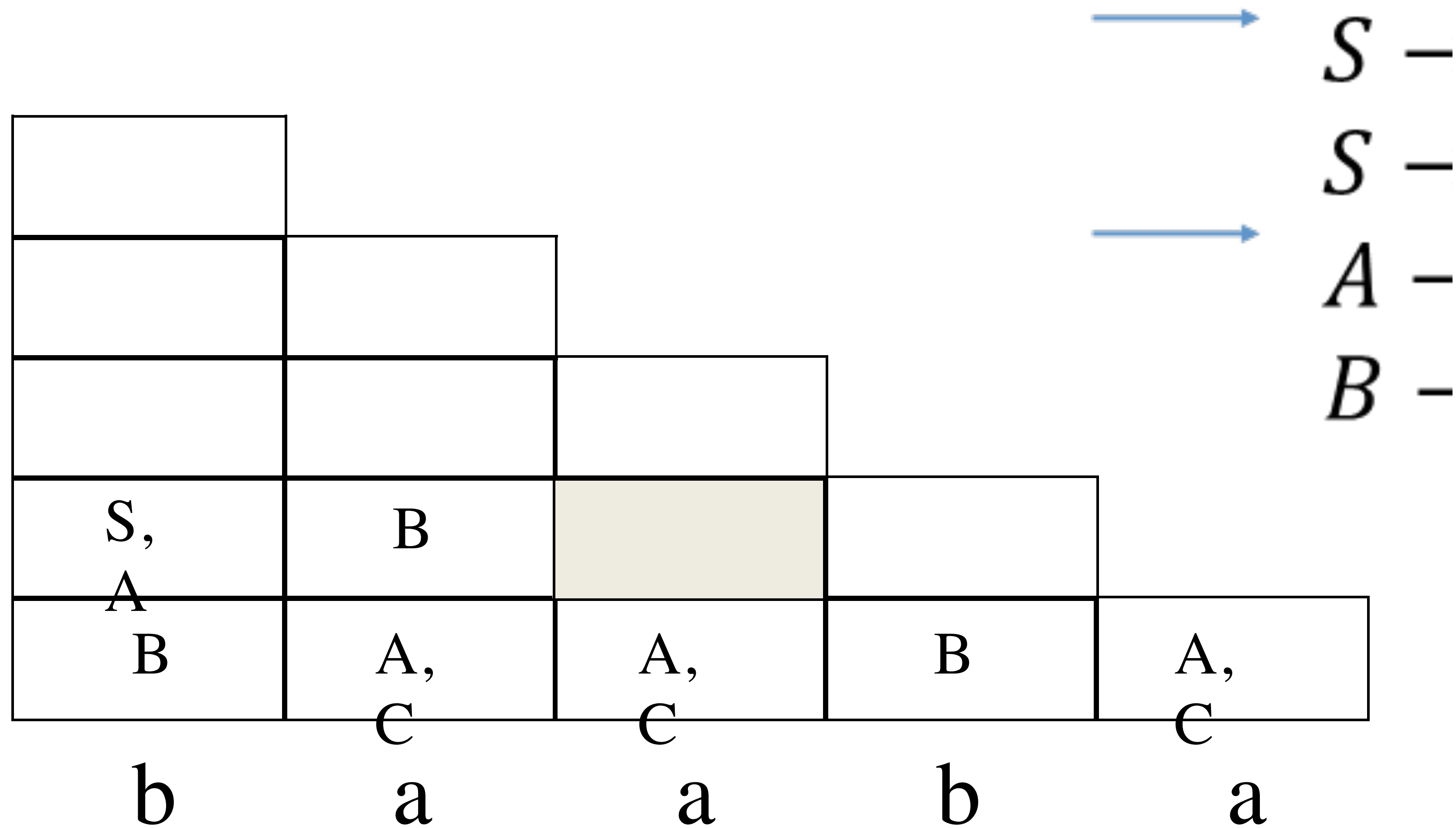
$S -$

$A -$

$B -$

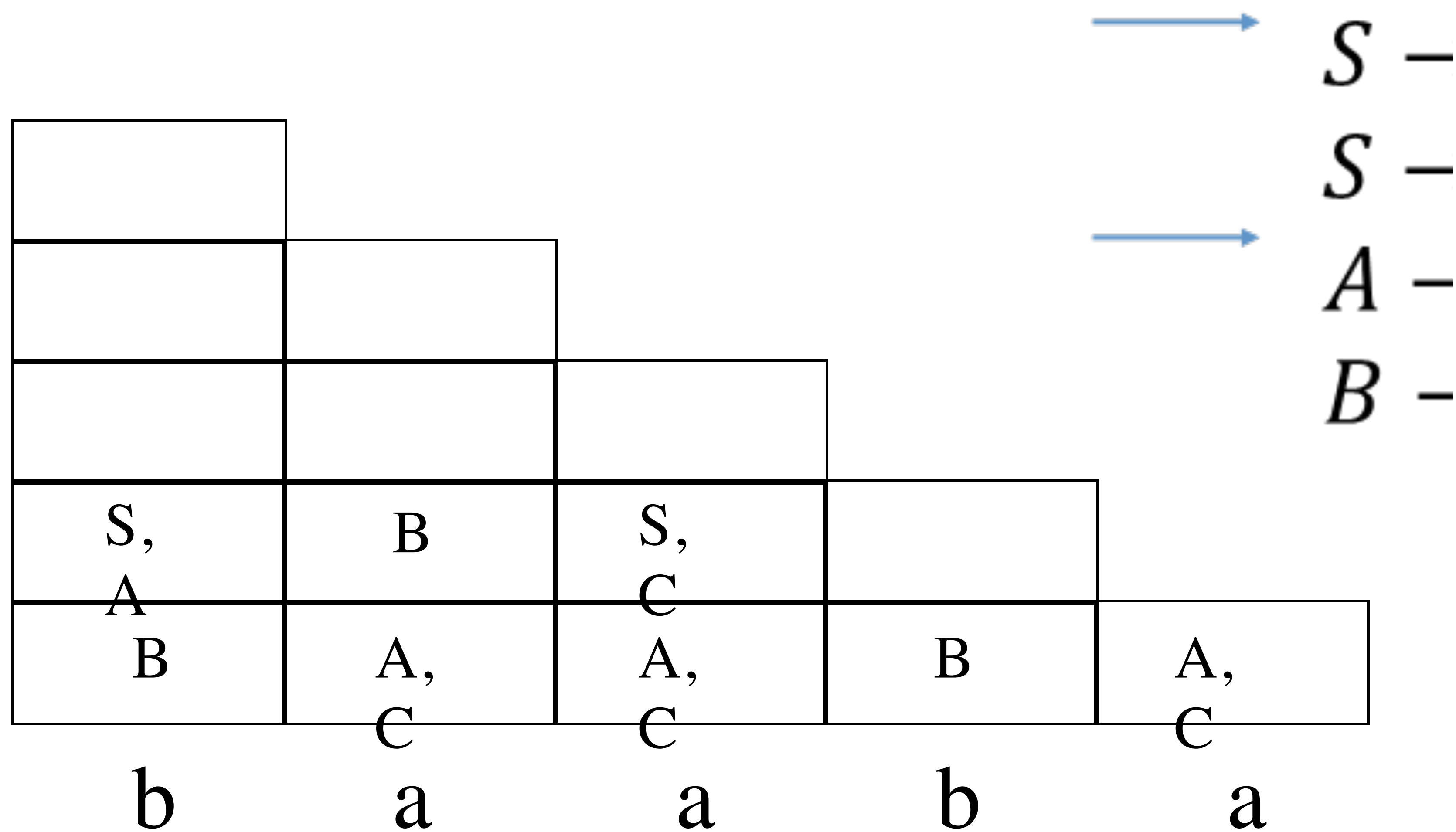
- Possible productions: A A, A C, C A, C C
- One rule

CYK example



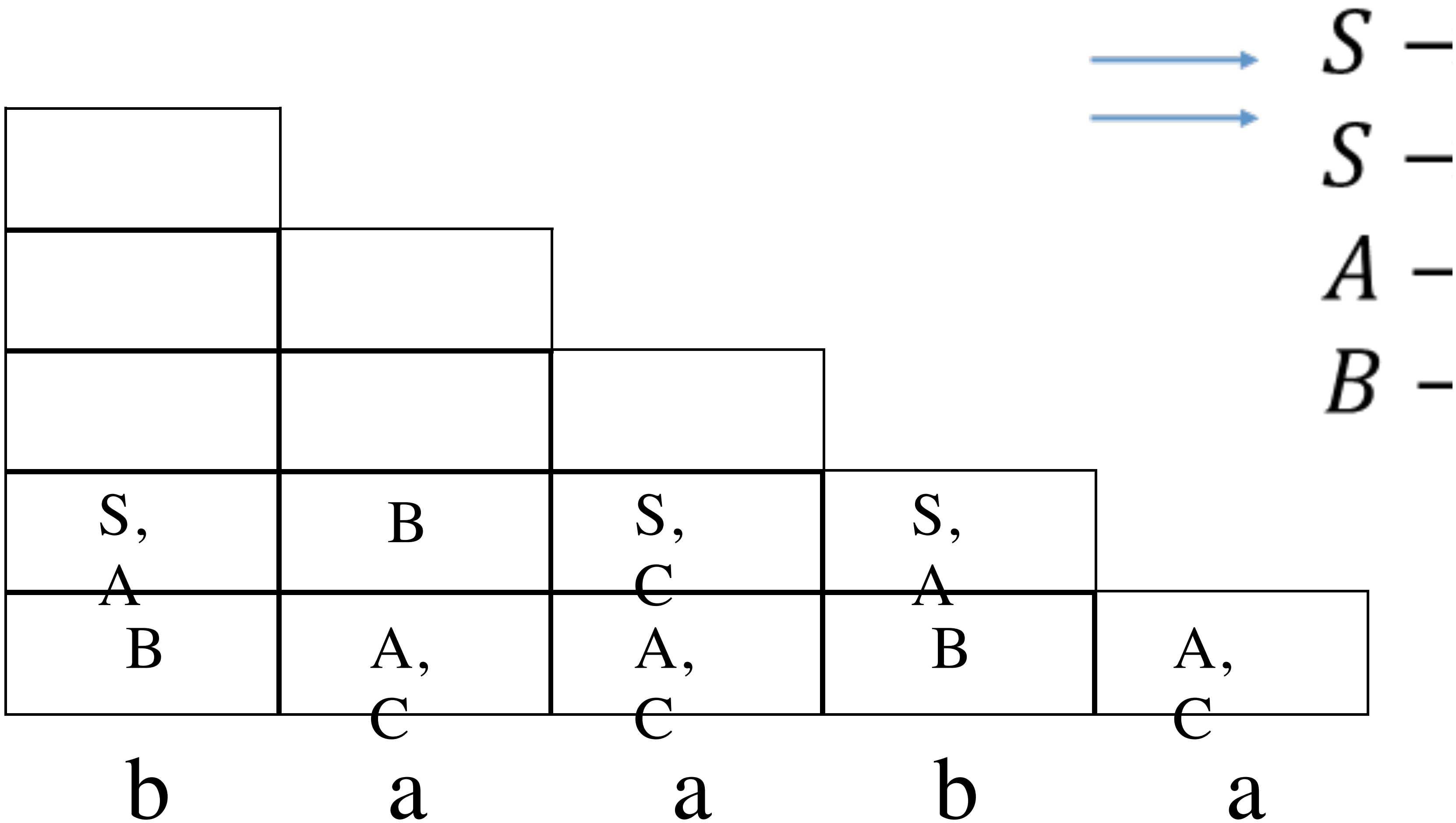
- Possible productions: A B, C B
- Two rules (note both produce A B)

CYK example



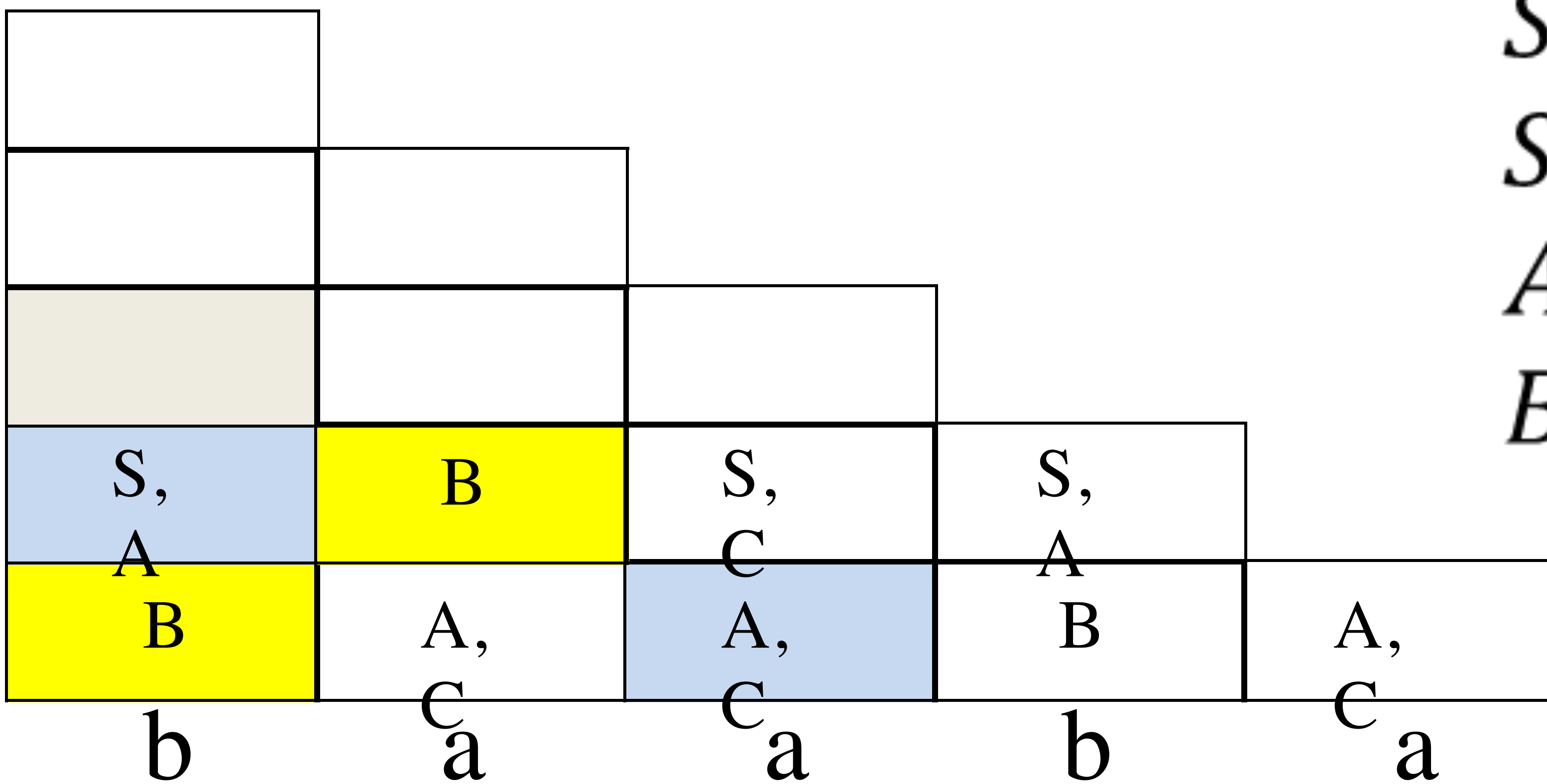
- Possible productions: A B, C B
- Two rules (note both produce A B)

CYK example



- Possible productions: B A, B C
- Two rules

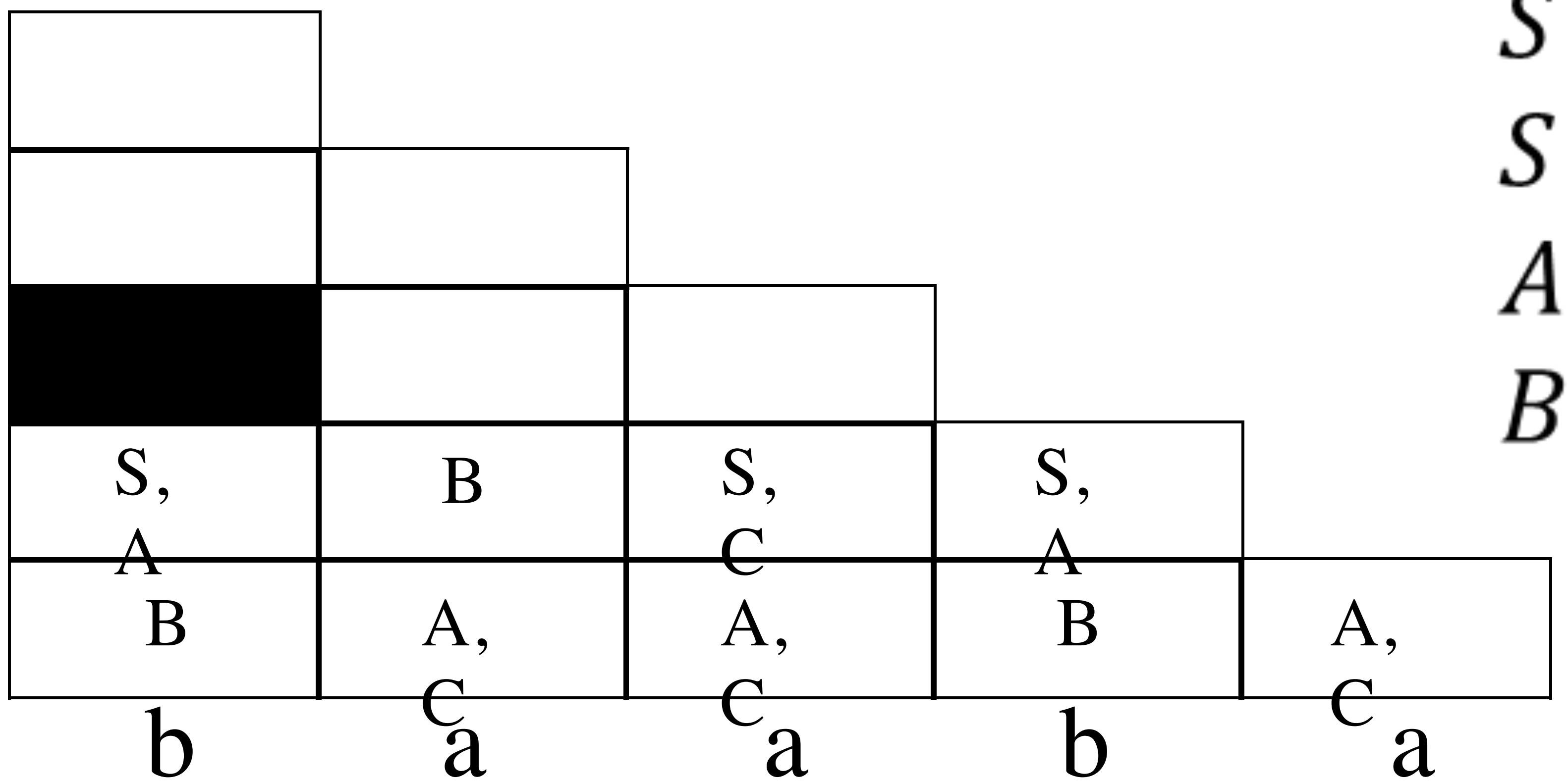
CYK example



$S -$
$S -$
$A -$
$B -$

- Two combinations that work
- Possible productions:
  - B B
  - S A, S C, A A, A C

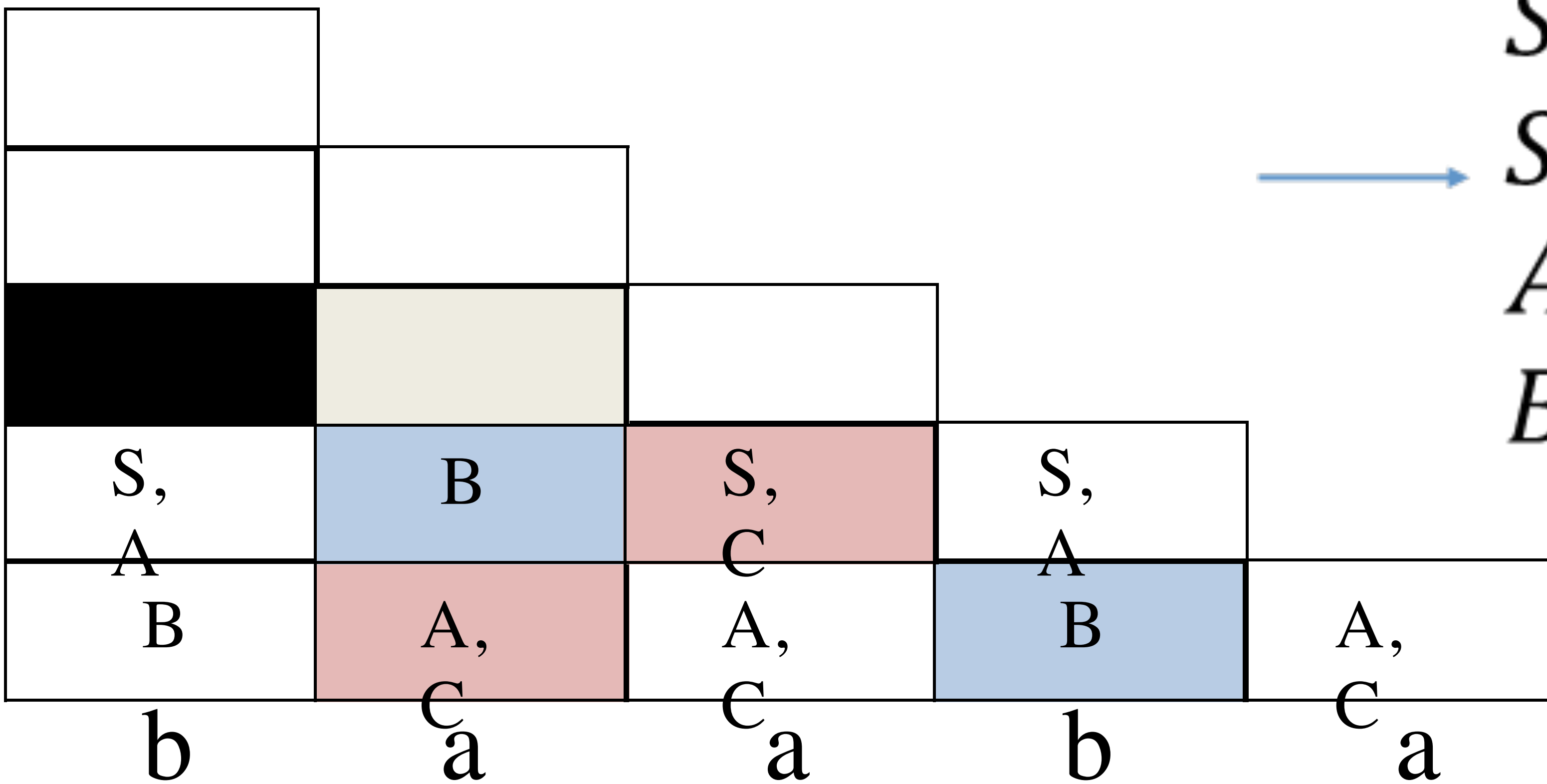
CYK example



$S –$
$S –$
$A –$
$B –$

- Two combinations that work
- Possible productions:
  - B B
  - S A, S C, A A, A C

CYK example

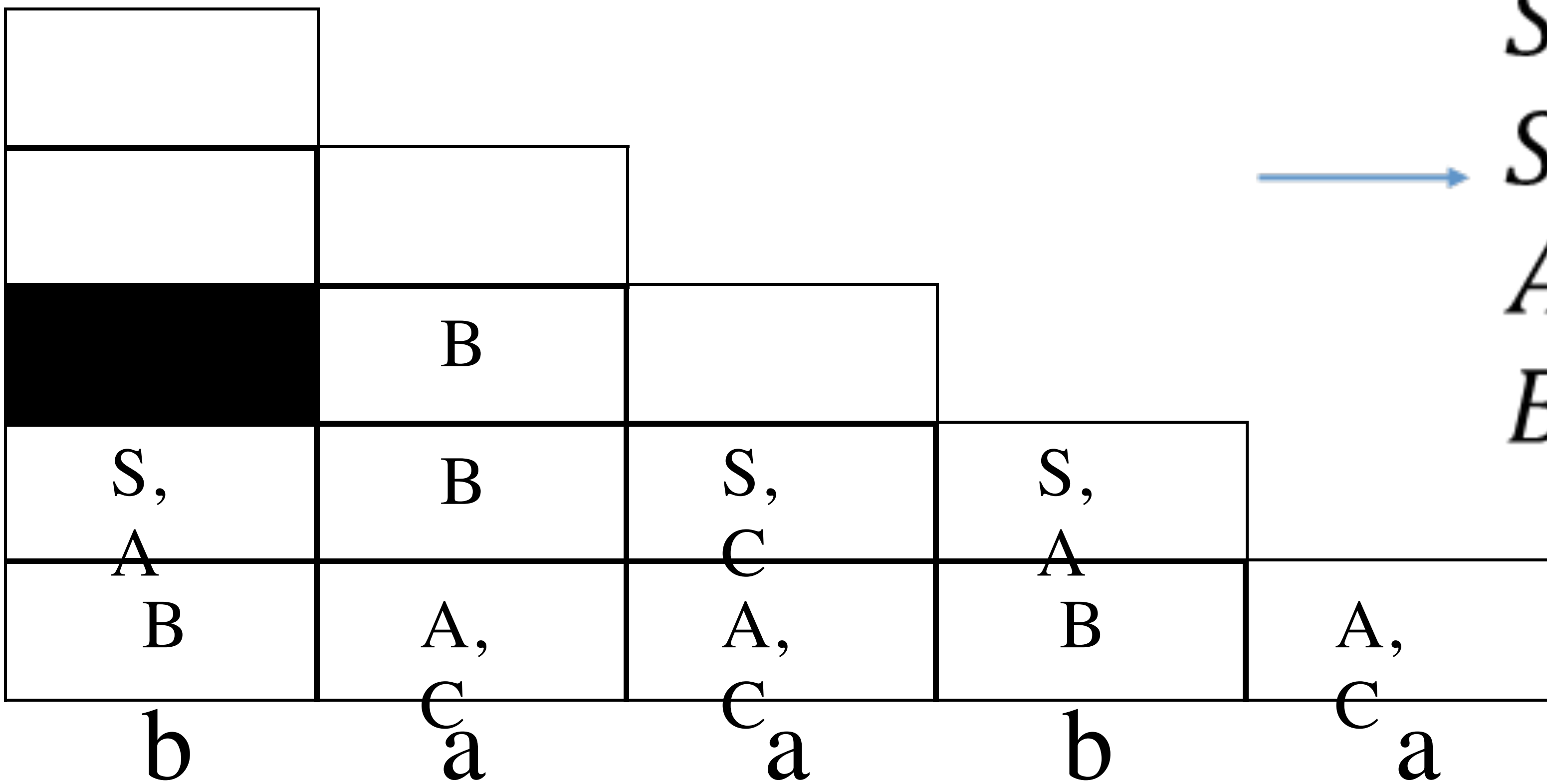


$S \rightarrow$
$S \rightarrow$
$A \rightarrow$
$B \rightarrow$

- Possible productions:
  - AS, AC, CS, CC
  - B B

  Only one rule

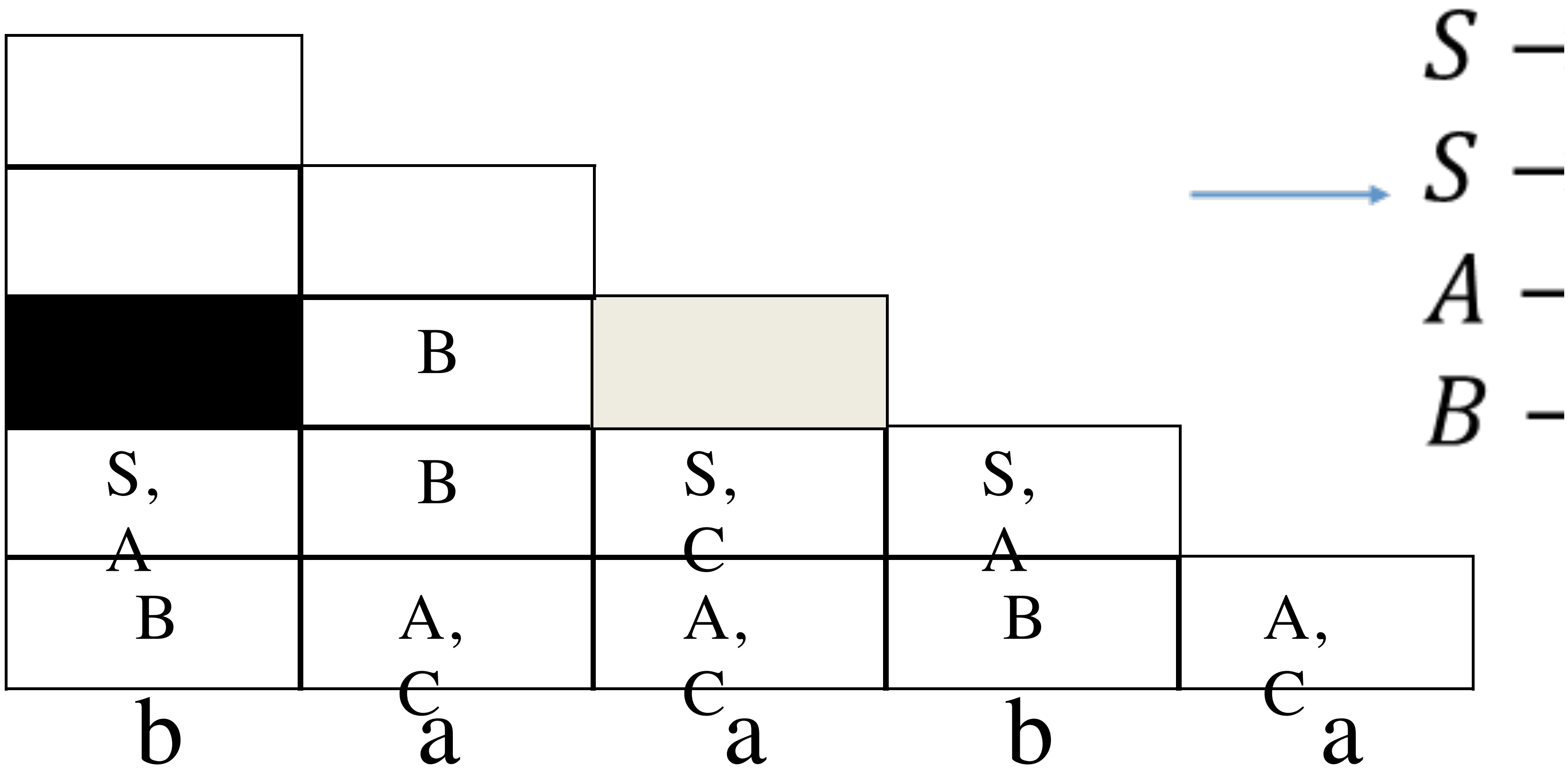
CYK example



$S -$
$S -$
$A -$
$B -$

- Possible productions:
  - AS, AC, CS, CC
  - B B
  - Only one rule

CYK example



- Possible productions:
  - AS, AA, CS, CA
  - SA, SC, CA, CC
  - Only one rule

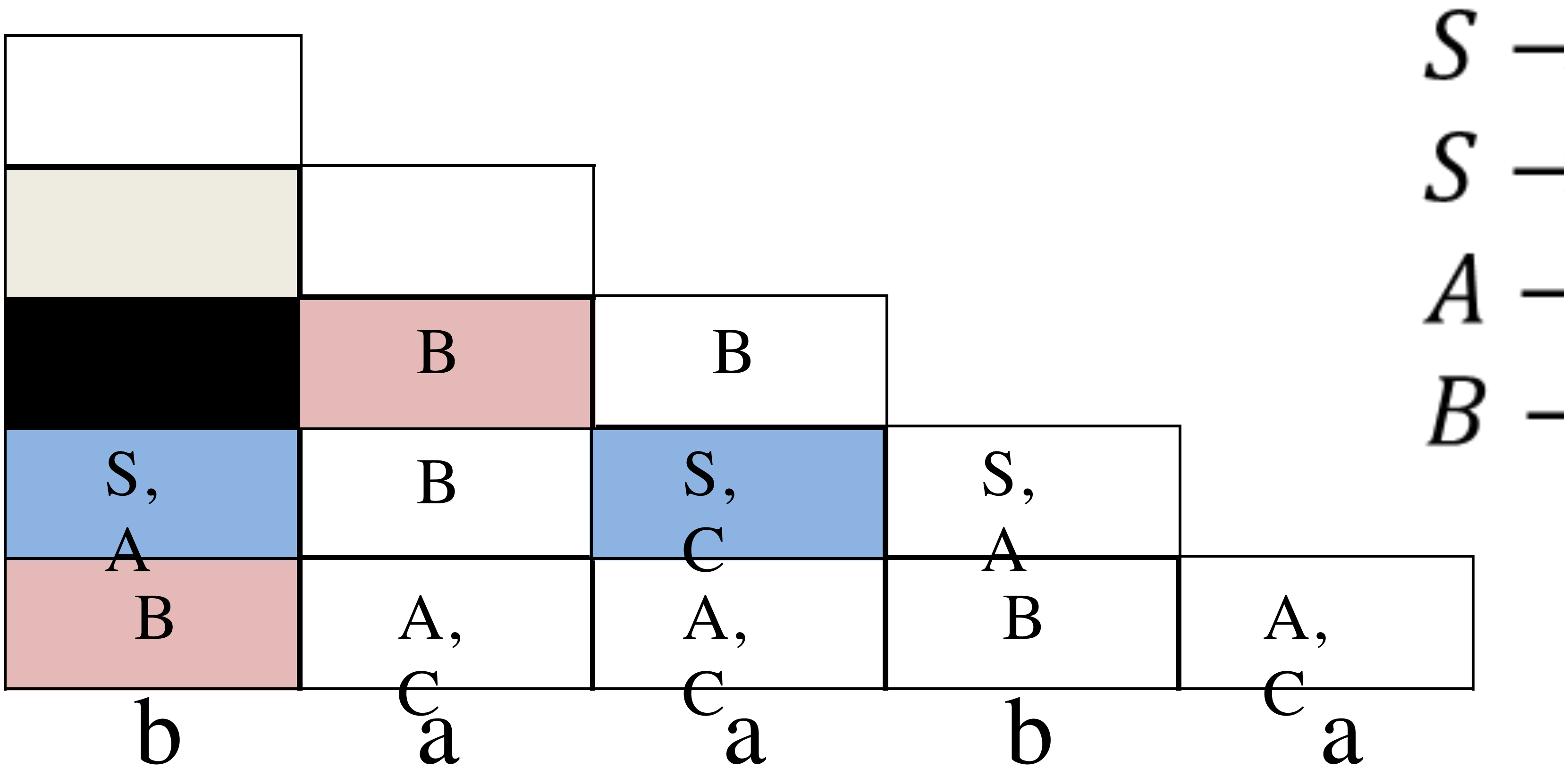
CYK example

	B	B		
S, A	B	S, C	S, A	
B	A, C	A, C	B	A, C
b	a	a	b	a

$S \rightarrow$
$S \rightarrow$
$A \rightarrow$
$B \rightarrow$

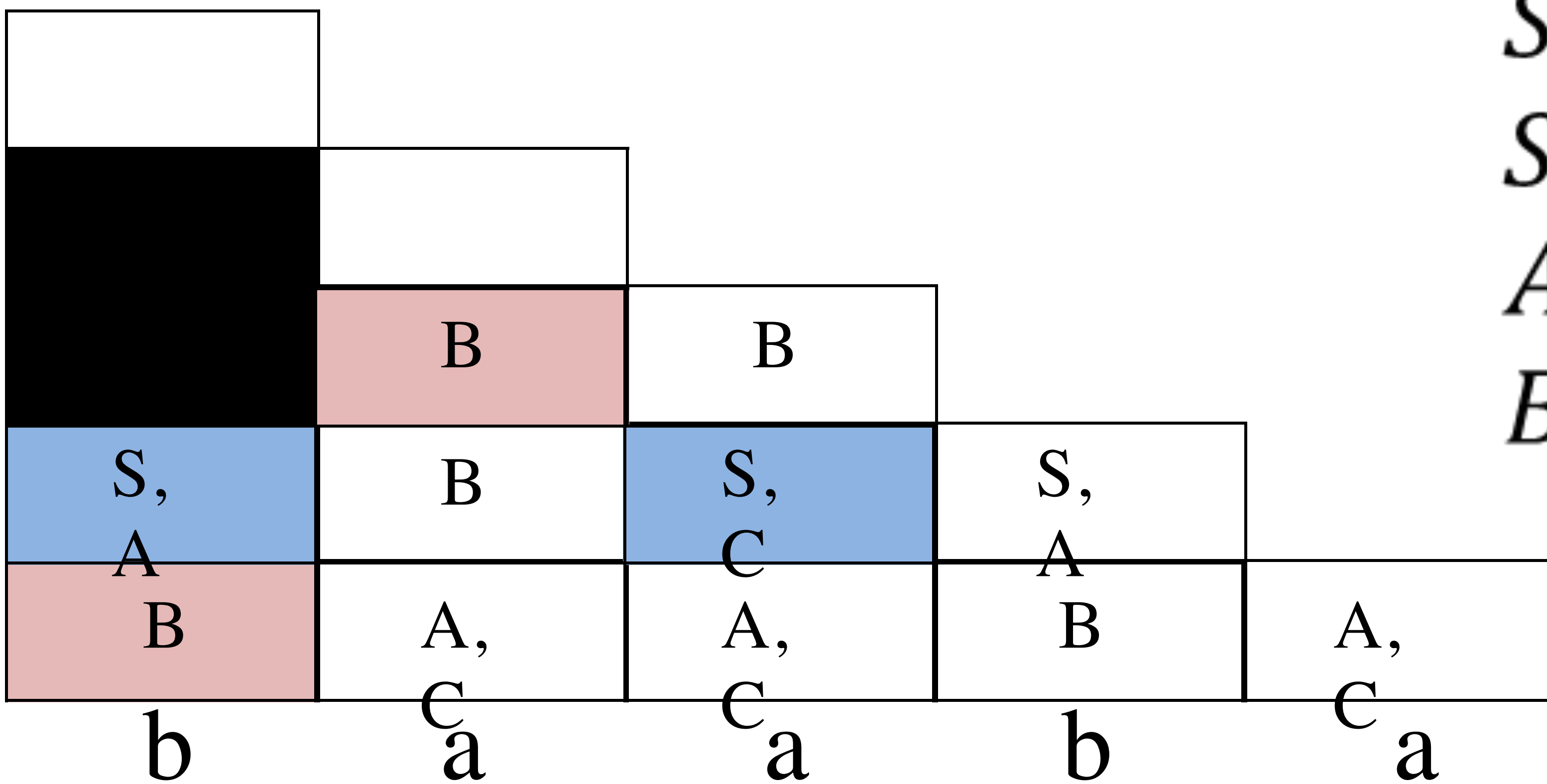
- Possible productions:
  - AS, AA, CS, CA
  - SA, SC, CA, CC
  - Only one rule

CYK example



- Possible productions:
  - SS, SC, AS, AC
  - BB
  - No rule for any of these

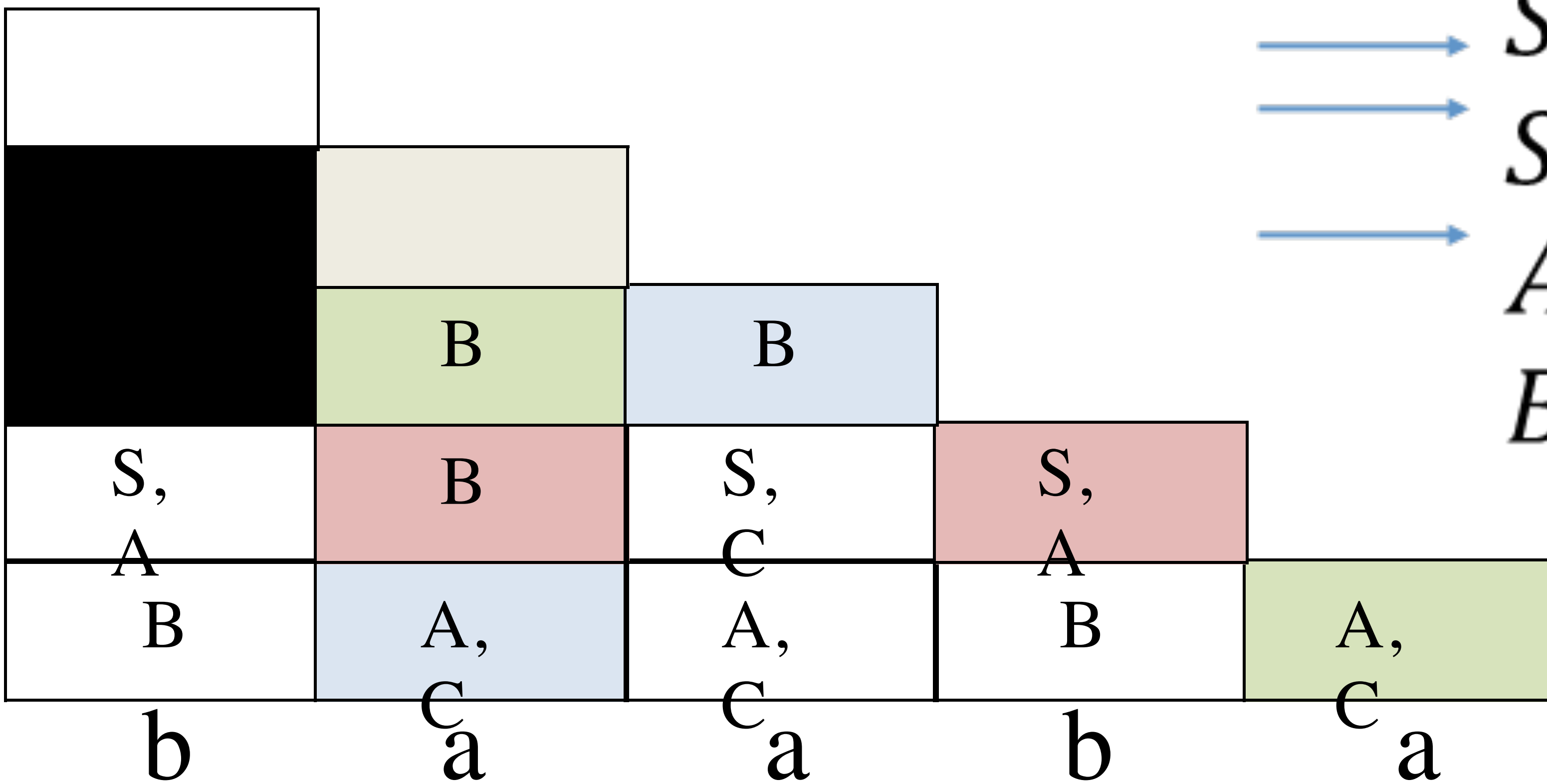
CYK example



S –
S –
A –
B –

- Possible productions:
  - SS, SC, AS, AC
  - BB
  - No rule for any of these

CYK example



S –

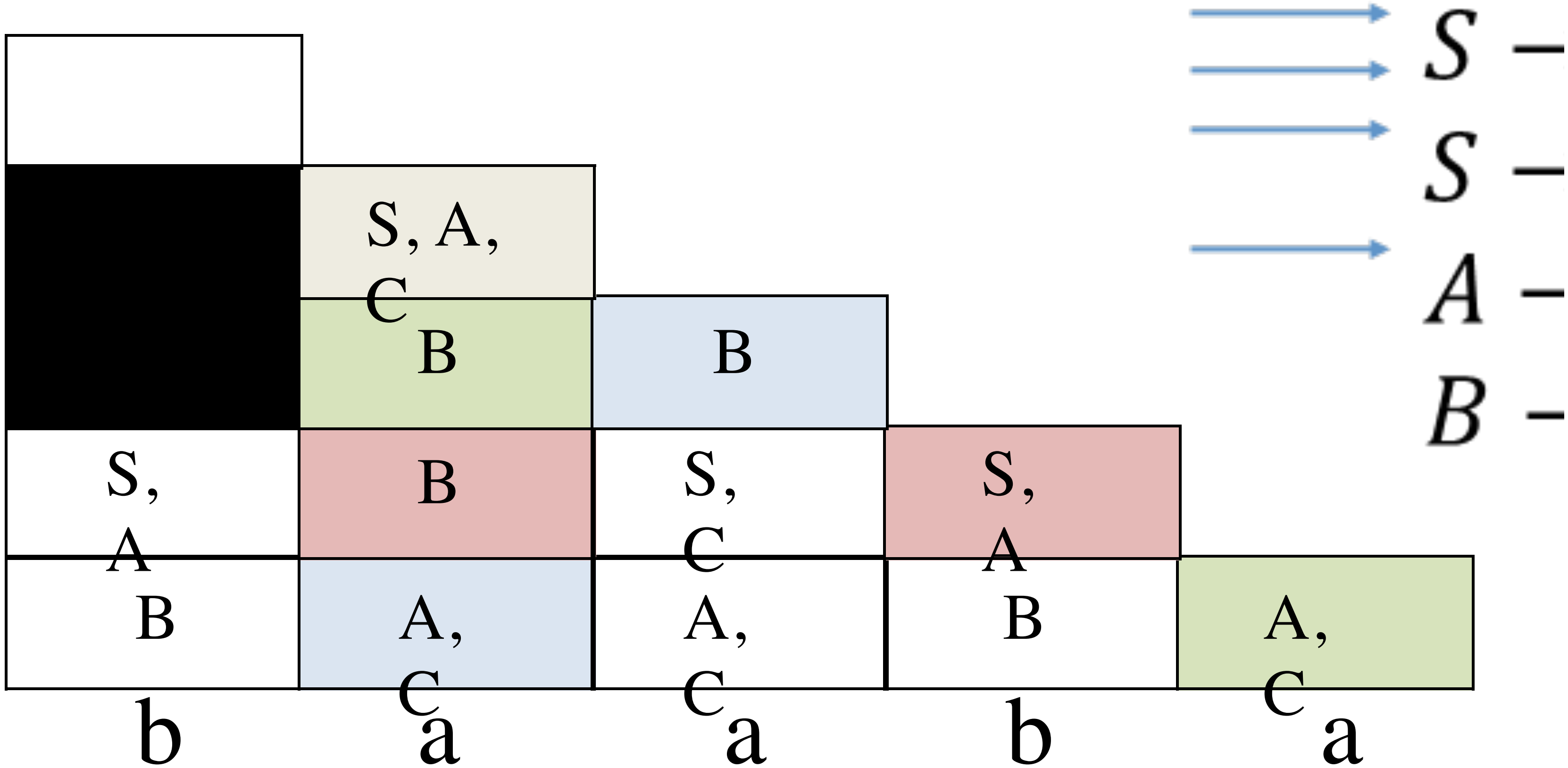
S –

A –

B –

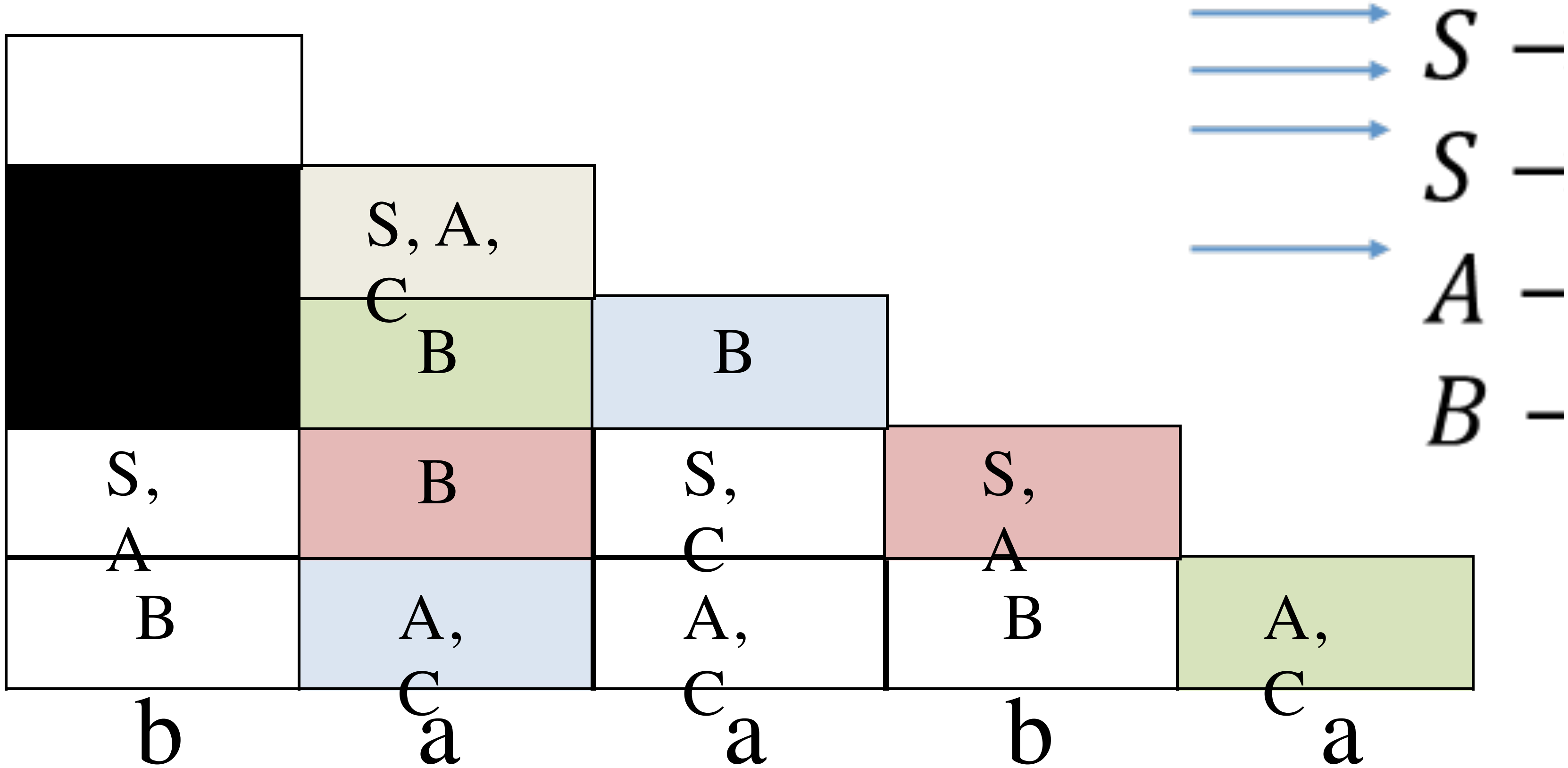
- Possible productions:
  - AB, CB
  - BS, BA
  - BA, BC

CYK example



- Possible productions:
  - AB, CB
  - BS, BA
  - BA, BC

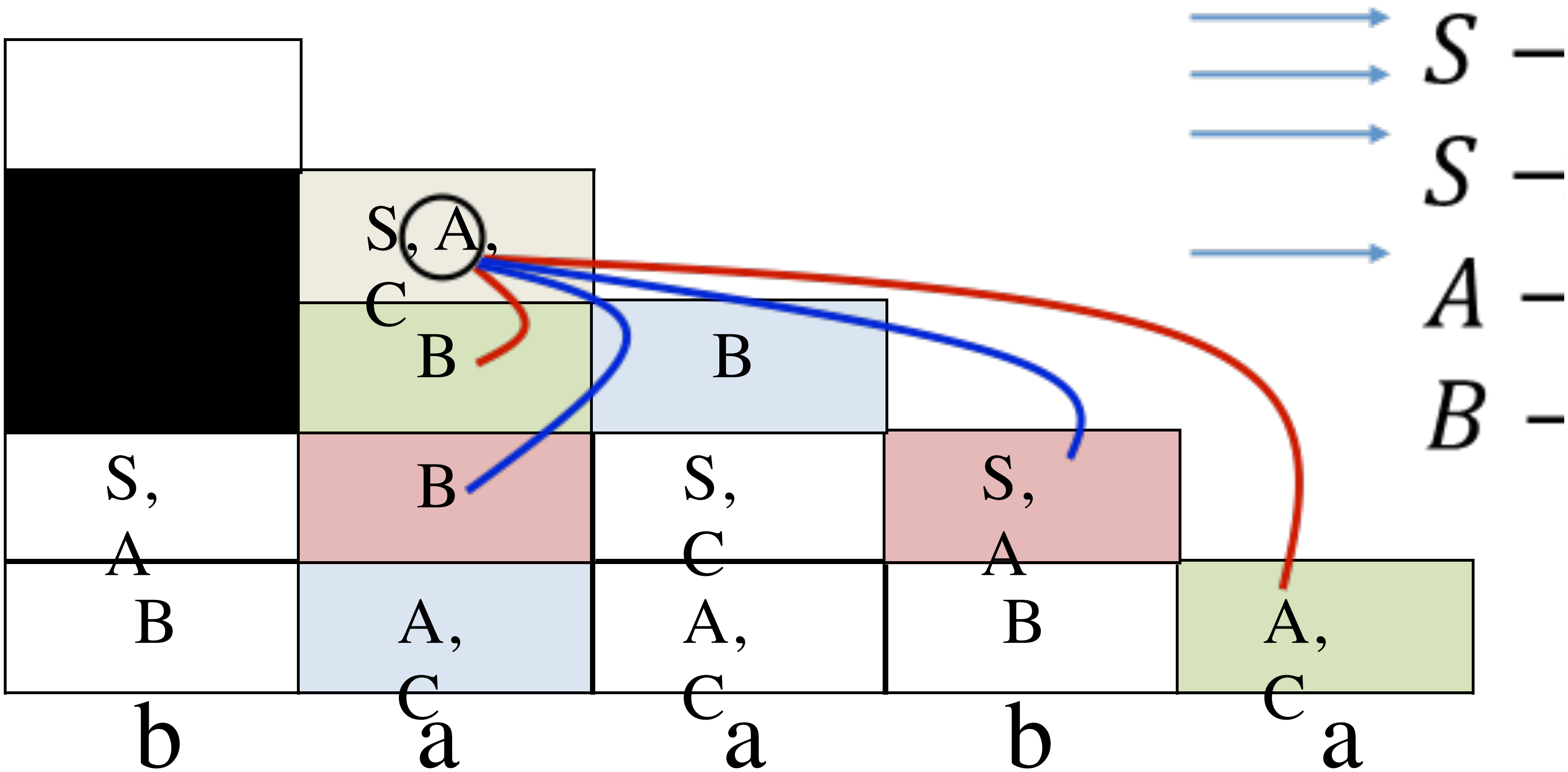
CYK example



- Possible productions:
  - AB, CB
  - BS, BA
  - BA, BC

IS THIS PARSE UNAMBIGUOUS?

CYK example

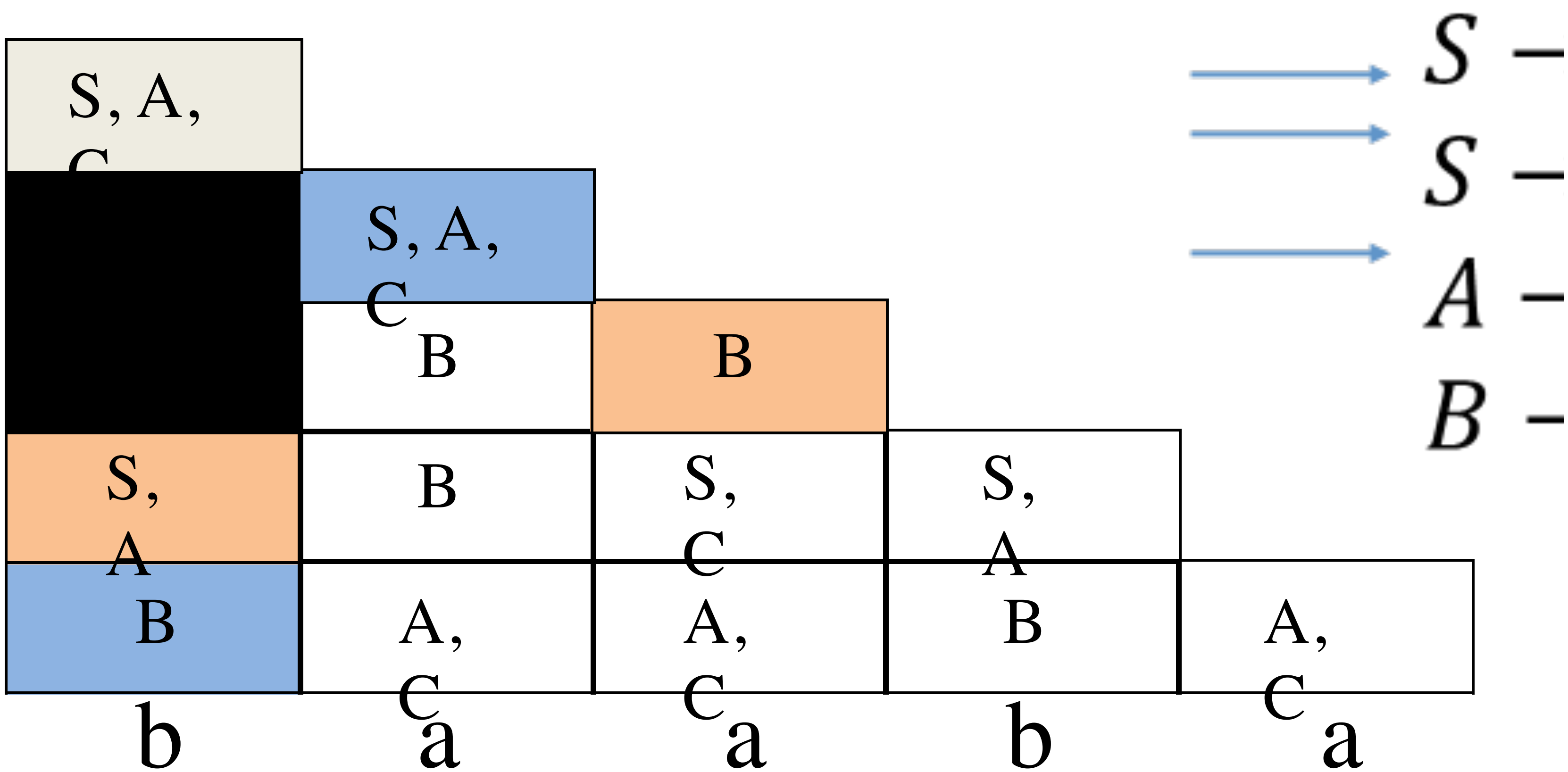


- Possible productions:
  - AB, CB
  - BS, BA
  - BA, BC

IS THIS PARSE UNAMBIGUOUS?

Possibly not… (can't be sure yet)

CYK example



- Possible productions:
  - BS, BA, BC
  - SB, AB
  - Three rules apply!

CYK example

S				
[redacted]	S, A, C			
[redacted]	B	B		
S, A	B	S, C	S, A	
B	A, C	A, C	B	A, C
b	a	a	b	a

$S -$
$S -$
$A -$
$B -$

- Check: Does the top box have S
  - Remove other entries

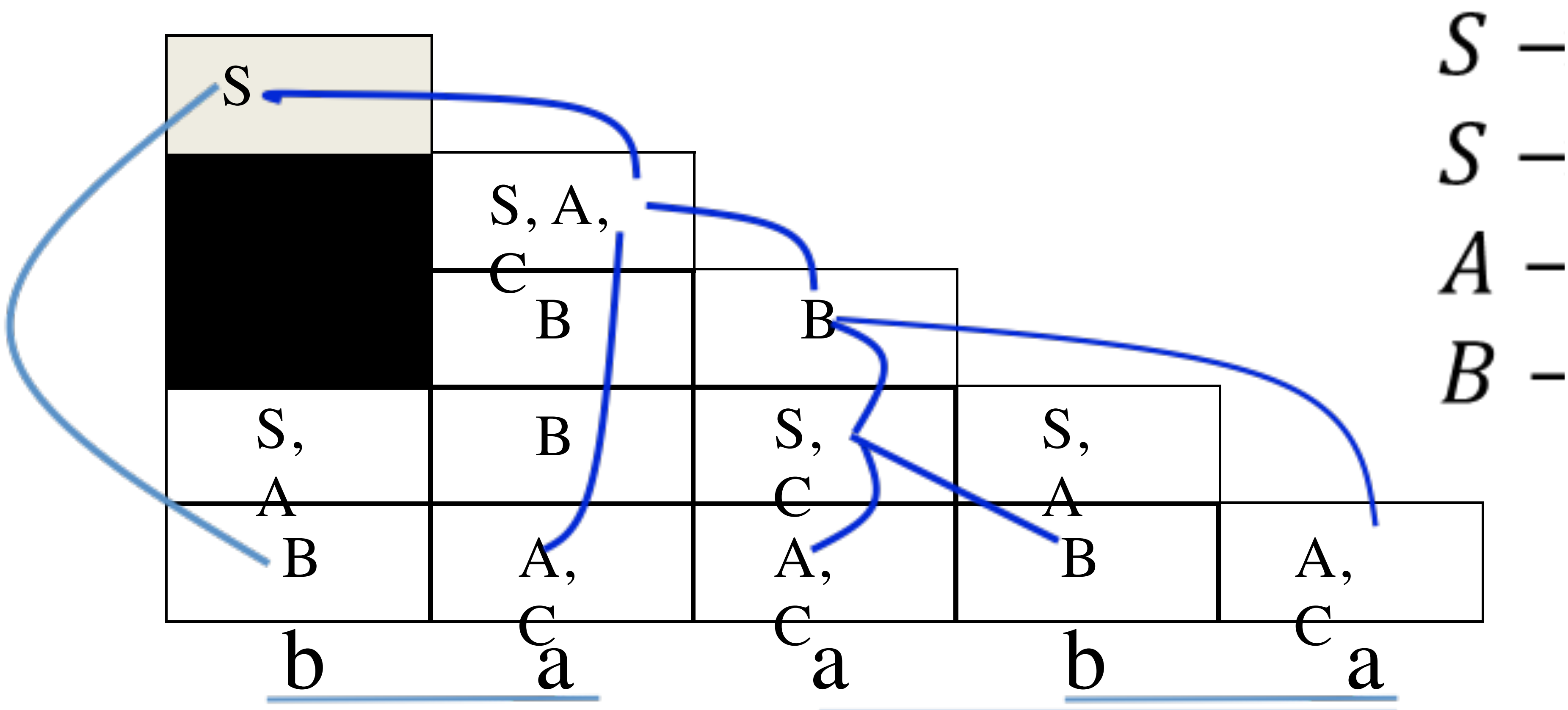
String does belong to the language

But how to find constituents?

- Need the parse tree for this

- At each box,
  - For each stored NT
    - keep track of not just the non-terminals, but the child nodes
- Forward trace from root to find the parse tree
  - The parse tree provides the constituents

CYK example



- Resulting parse
  - Constituents can be found from it
- Tracing the parse tree is possible because this is an

CYK example

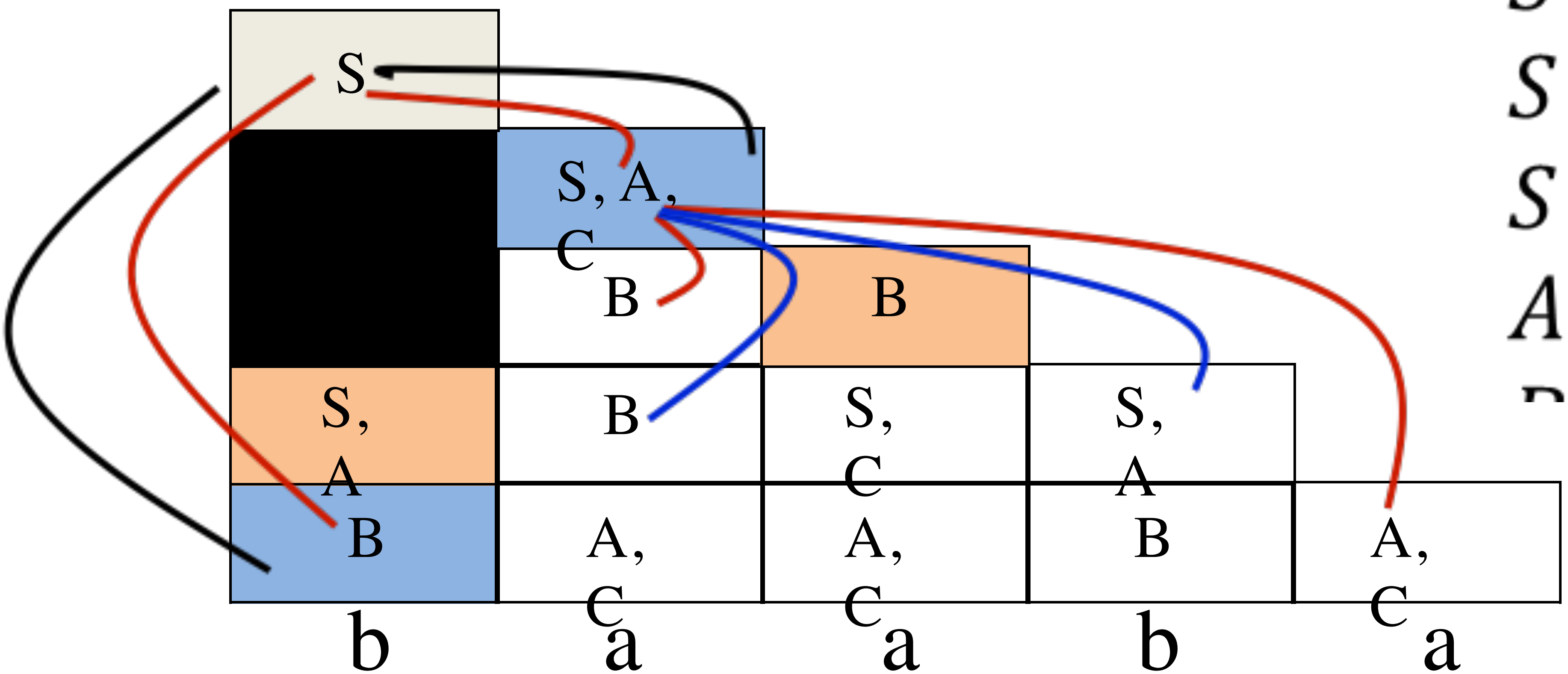
	S, A, C			
	B	B		
S, A	B	S, C	S, A	
B	A, C	A, C	B	A, C
b	a	a	b	a

$S -$
$S -$
$S -$
$A -$

•

- Now add another rul

CYK example



$S -$

$S -$

$S -$

$A -$

- Possible productions:
  - BS, BA, BC
  - SB, AB
- Multiple S rules apply

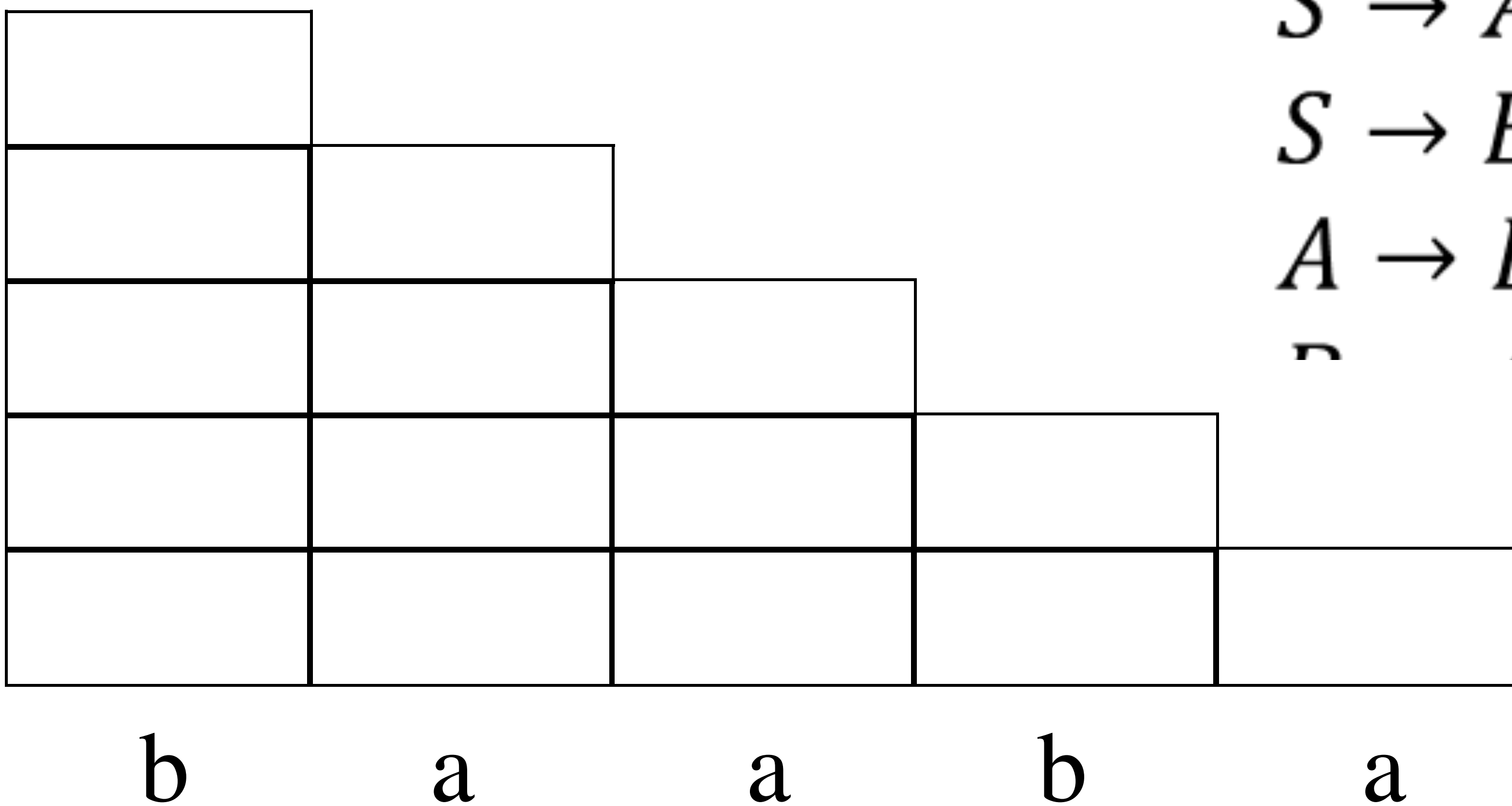
Not an unambiguous parse!

Many possible parses

How do we choose the best parse?

CYK with PCFG

$S \to B$
$S \to A$
$S \to B$
$A \to B$



- Rules now have probabilities
  - Note, probabilities of all expansions of any specific NT sum to 1.0

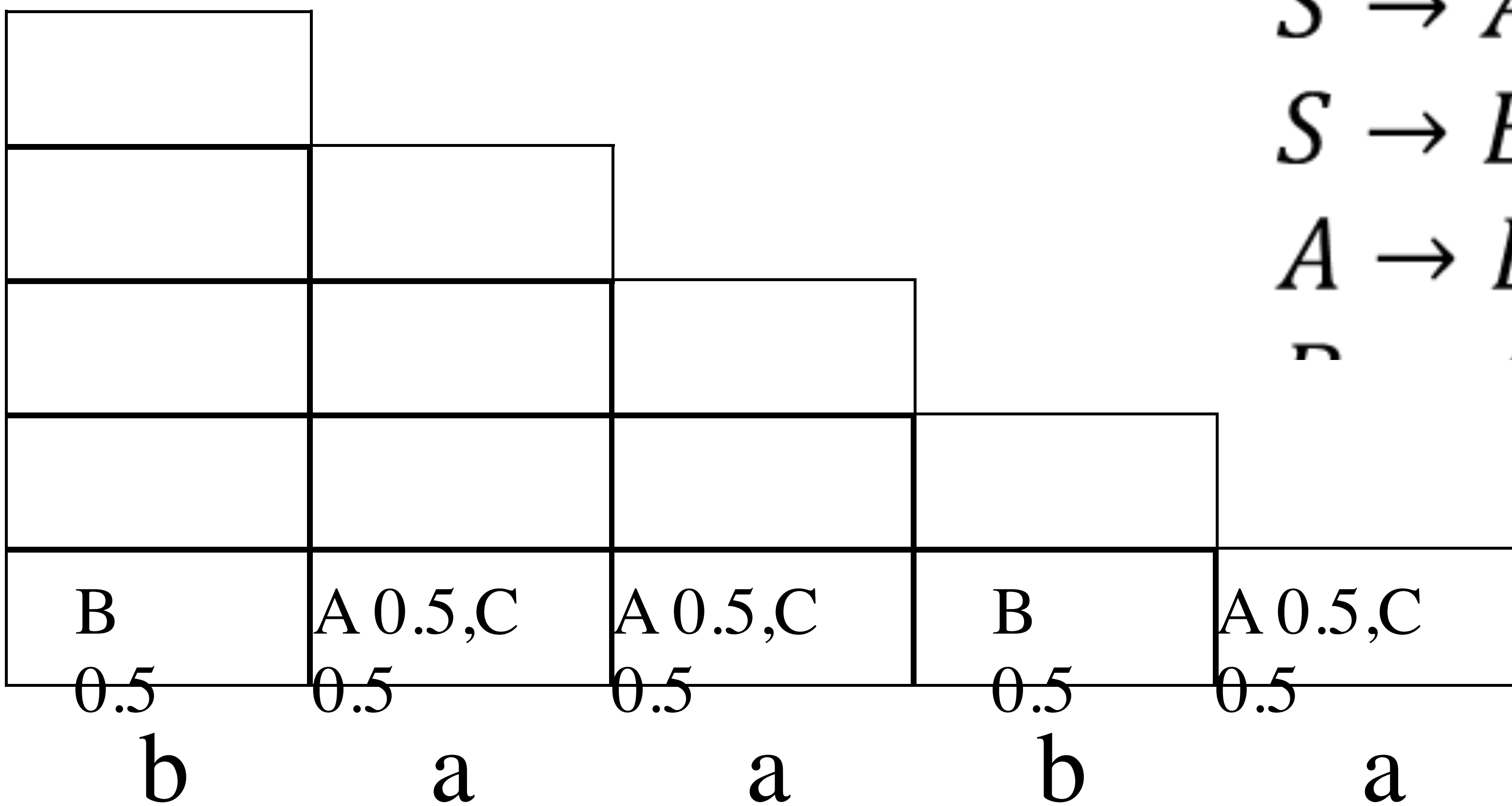
CYK example

$S \rightarrow B$

$S \rightarrow A$

$S \rightarrow B$

$A \rightarrow B$



- Keep track of probabilities of rules applied

CYK example

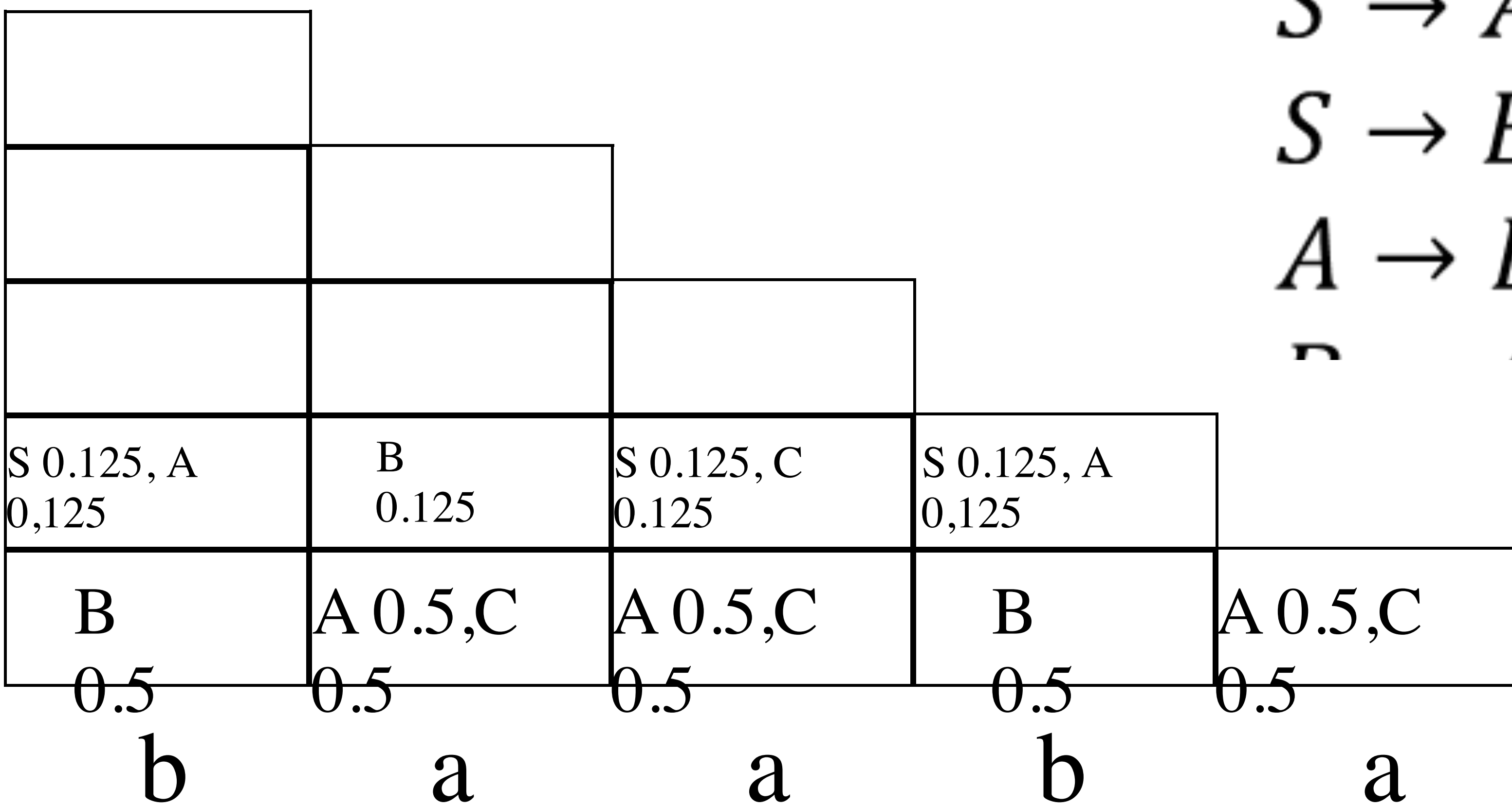
$S \rightarrow B$
$S \rightarrow A$
$S \rightarrow B$
$A \rightarrow B$

S 0.125, A 0,125				
B 0.5	A 0.5,C 0.5	A 0.5,C 0.5	B 0.5	A 0.5,C 0.5
b	a	a	b	a

- For each new rule inserted in table, multiply the probability of the rule by the probability of

CYK example

$S \to B$
$S \to A$
$S \to B$
$A \to B$



- For each new rule inserted in table, multiply the probability of the rule by the probability of

CYK example

$S \to B$
$S \to A$
$S \to B$
$A \to B$

0	B 1/32	B 1/32		
S 0.125, A 0,125	B 0.125	S 0.125, C 0.125	S 0.125, A 0,125	
B 0.5	A 0.5,C 0.5	A 0.5,C 0.5	B 0.5	A 0.5,C 0.5
b	a	a	b	a

- For each new rule inserted in table, multiply the probability of the rule by the probability of its children

CYK example

0	C 1/128			
0	B 1/32	B 1/32		
S 0.125, A 0,125	B 0.125	S 0.125, C 0.125	S 0.125, A 0,125	
B 0.5	A 0.5,C 0.5	A 0.5,C 0.5	B 0.5	A 0.5,C 0.5
b	a	a	b	a

$S \to B$

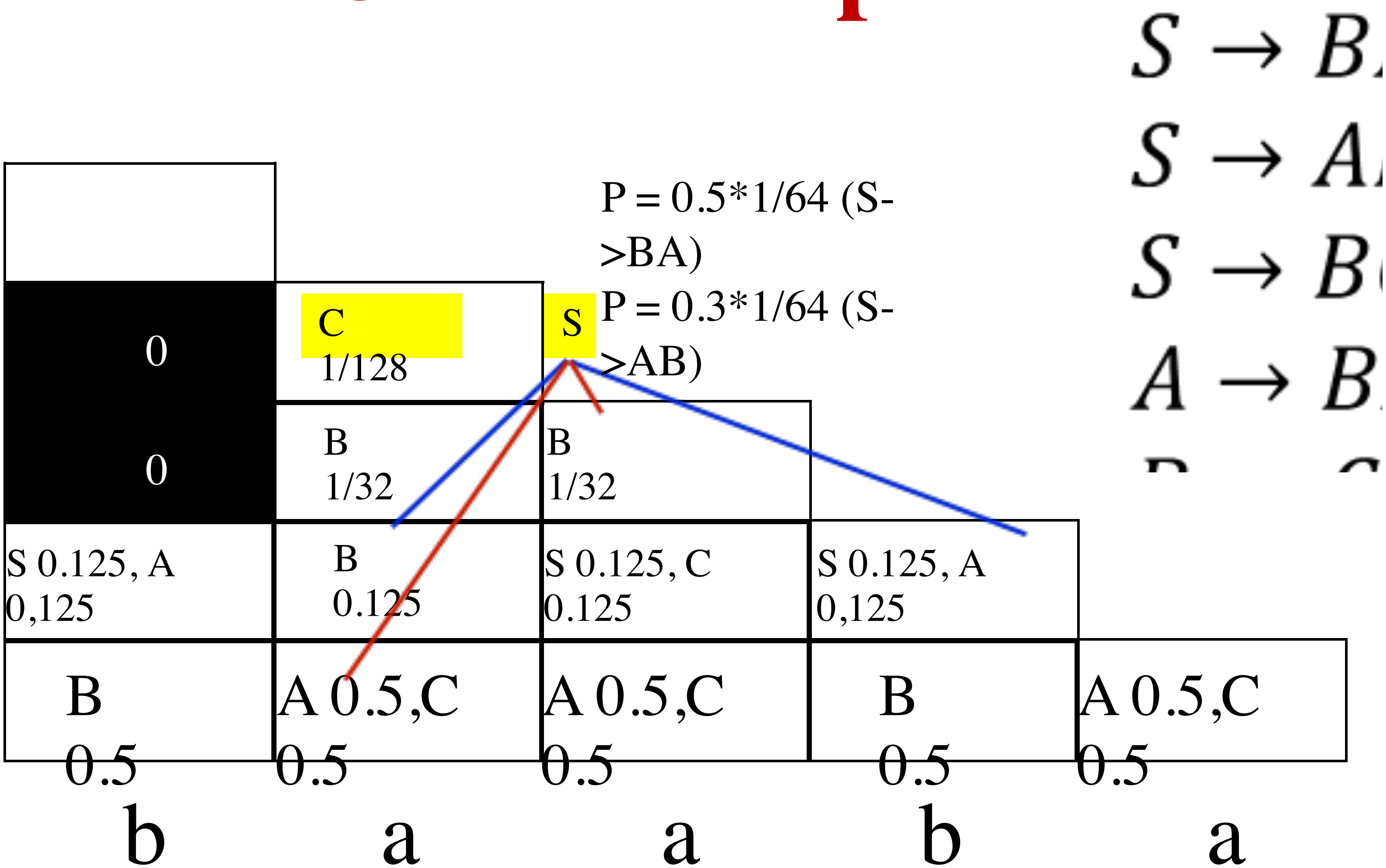
$S \to A$

$S \to B$

$A \to B$

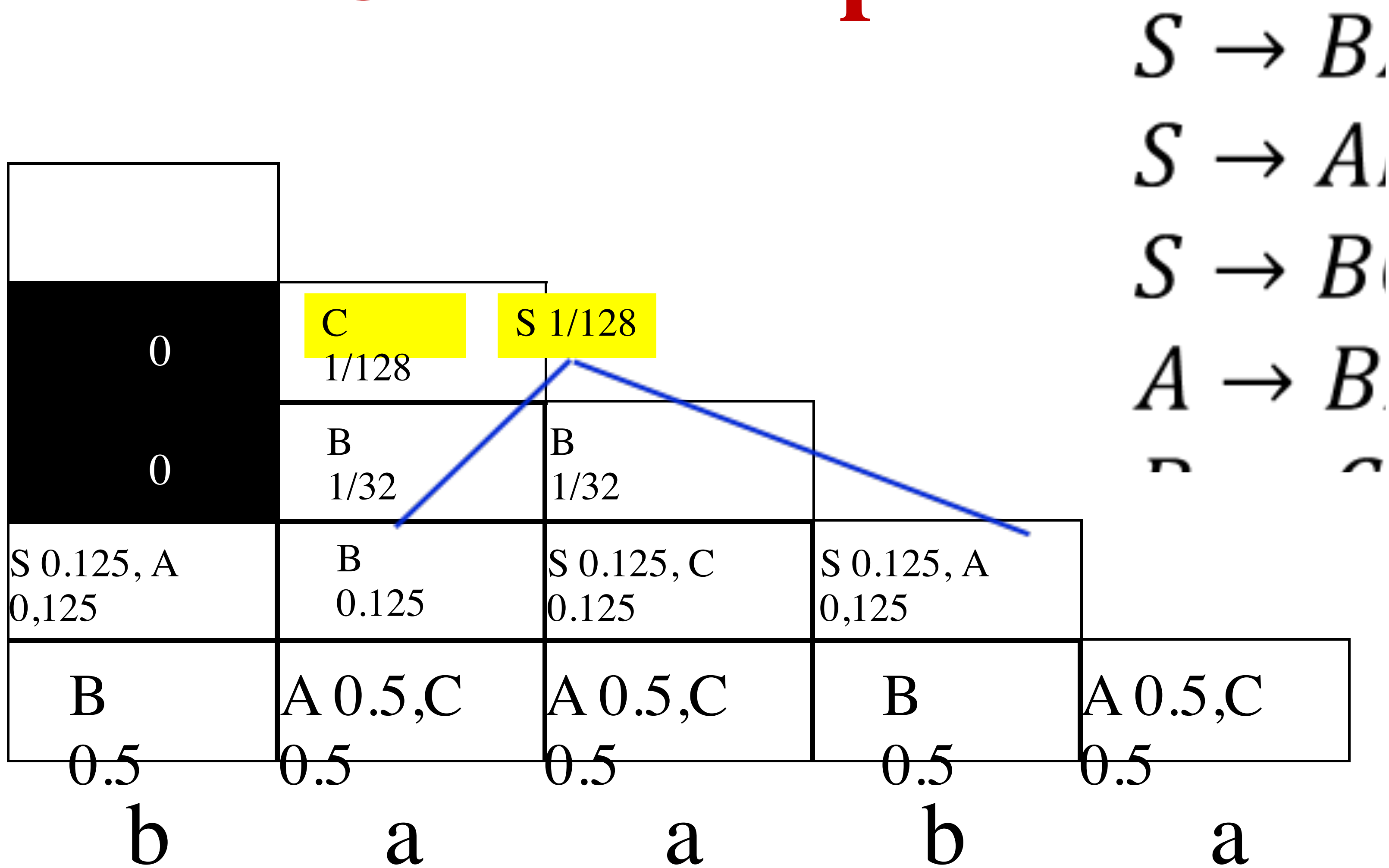
- For each new rule inserted in table, multiply the probability of the rule by the probability of

CYK example



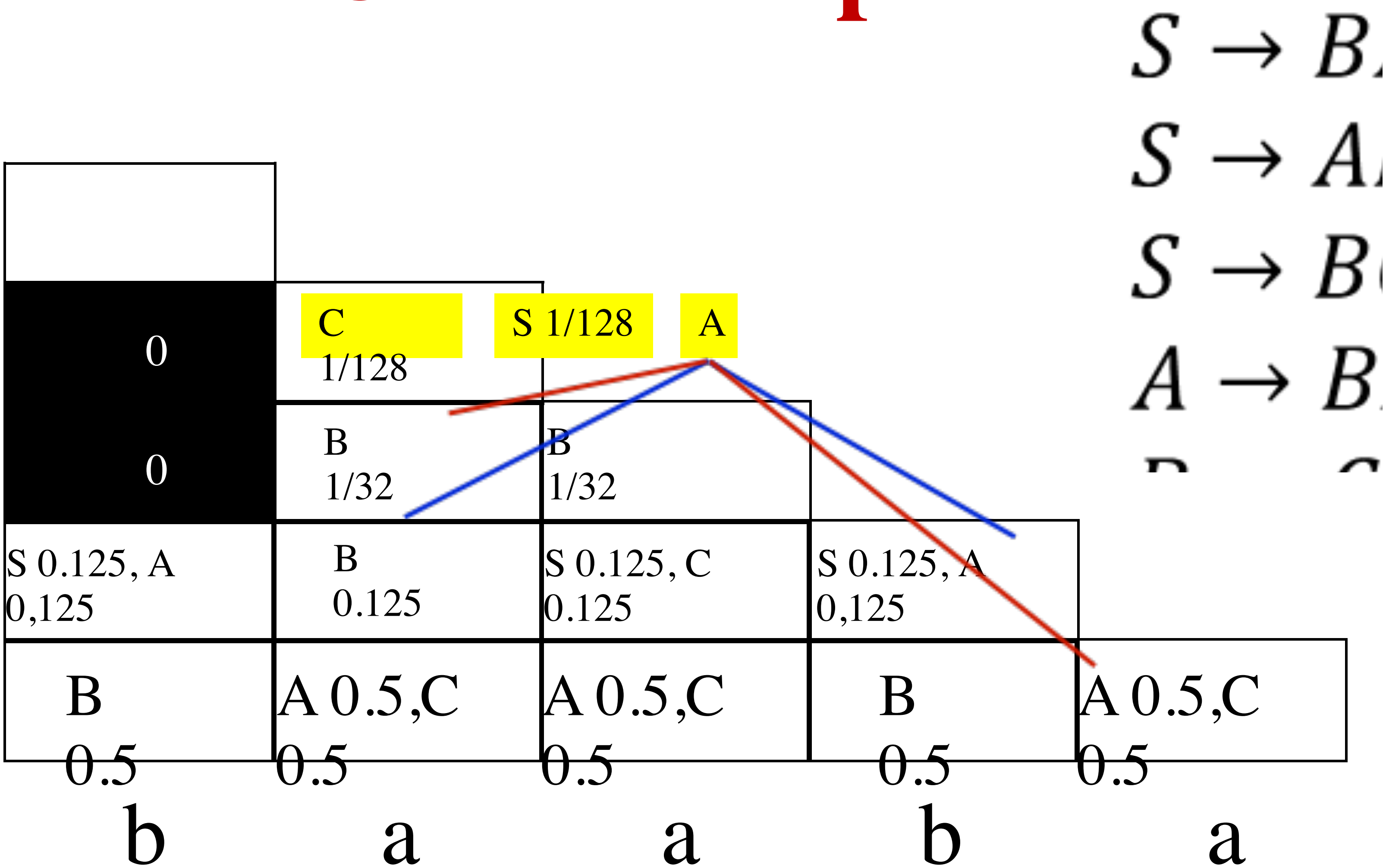
- When a rule has two or more possible productions, pick the most probable one

CYK example



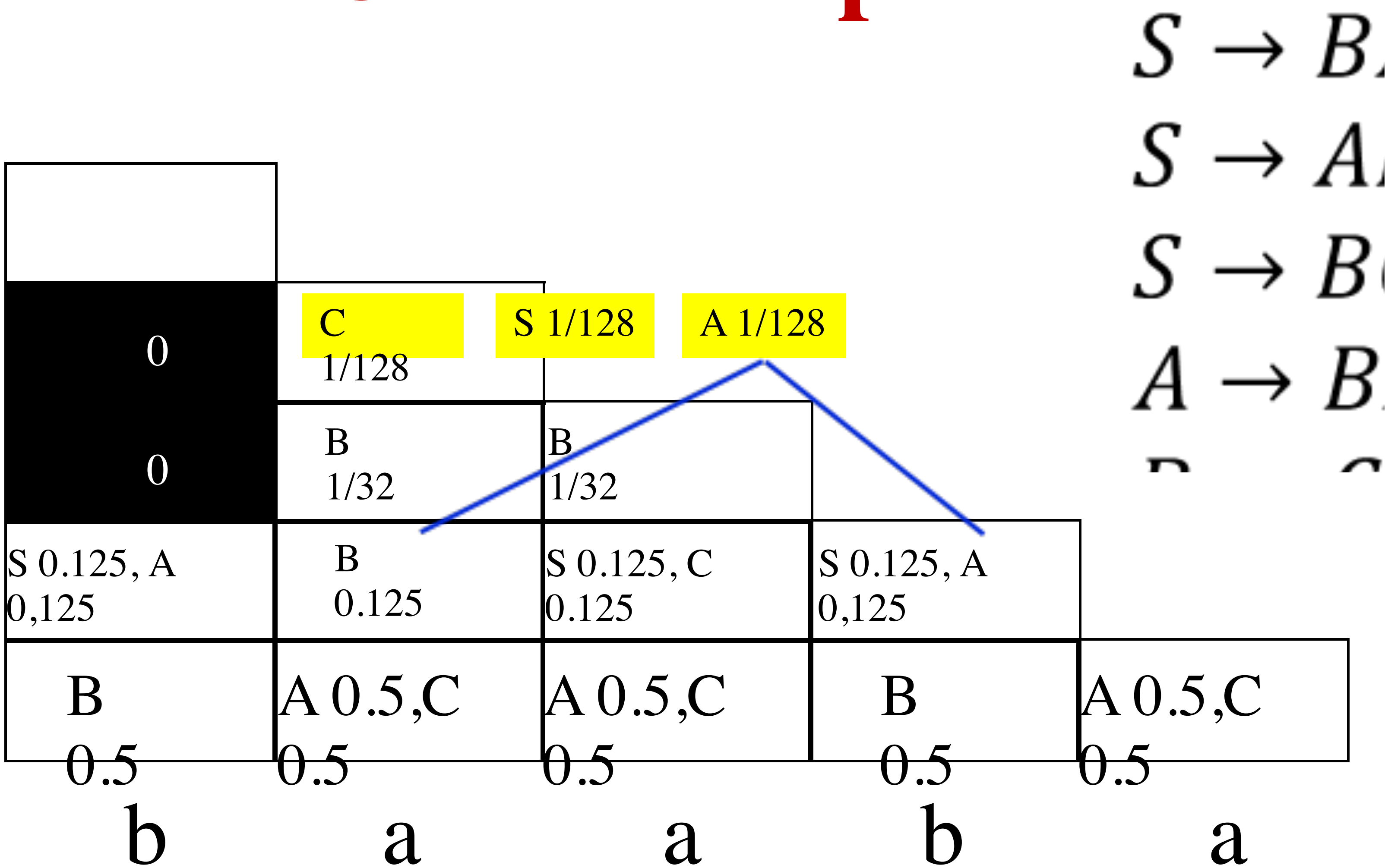
- When a rule has two or more possible productions, pick the most probable one

CYK example



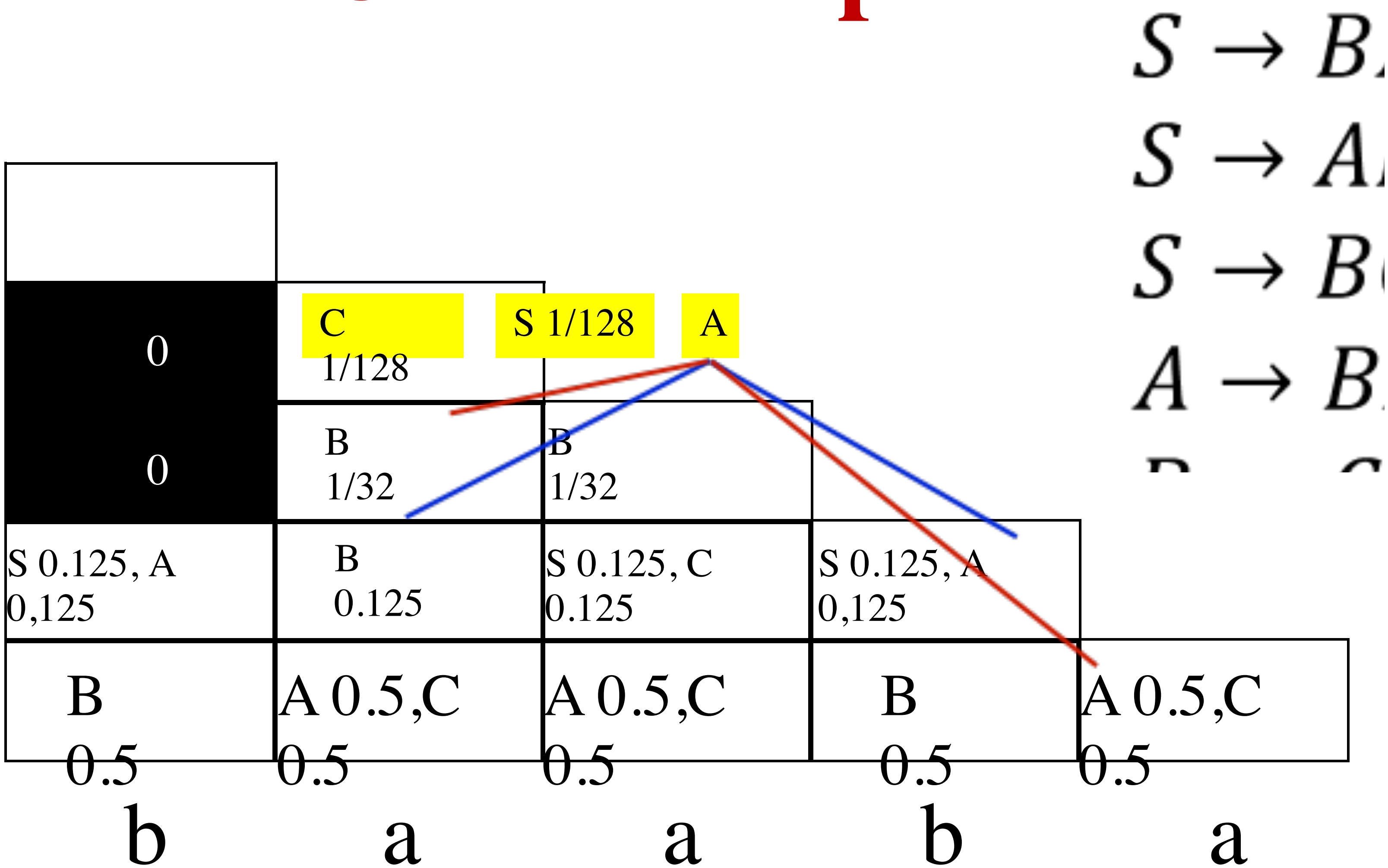
- When a rule has two or more possible productions, pick the most probable one

CYK example



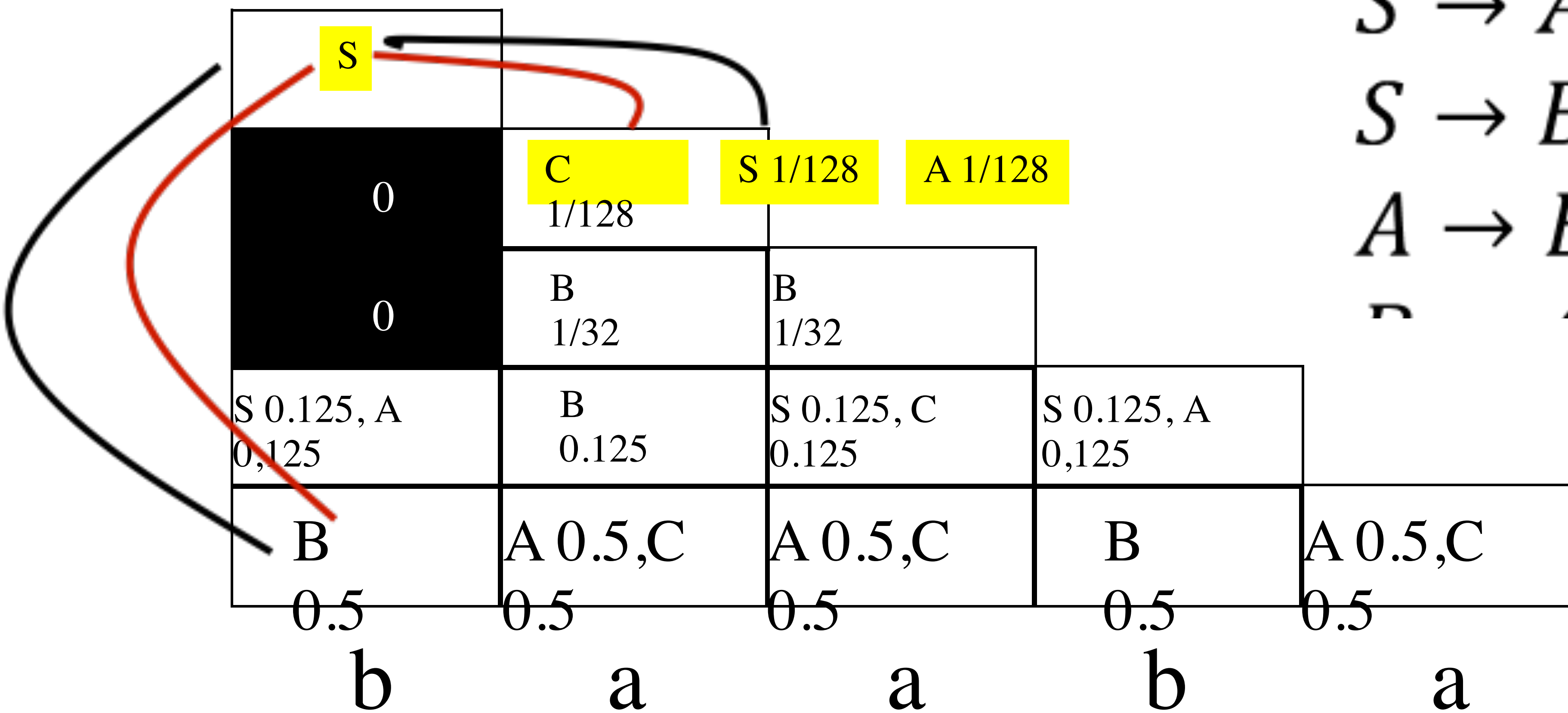
- When a rule has two or more possible productions, pick the most probable one

CYK example



- Note: Competition happens only between different expansions of the *same non-terminal*

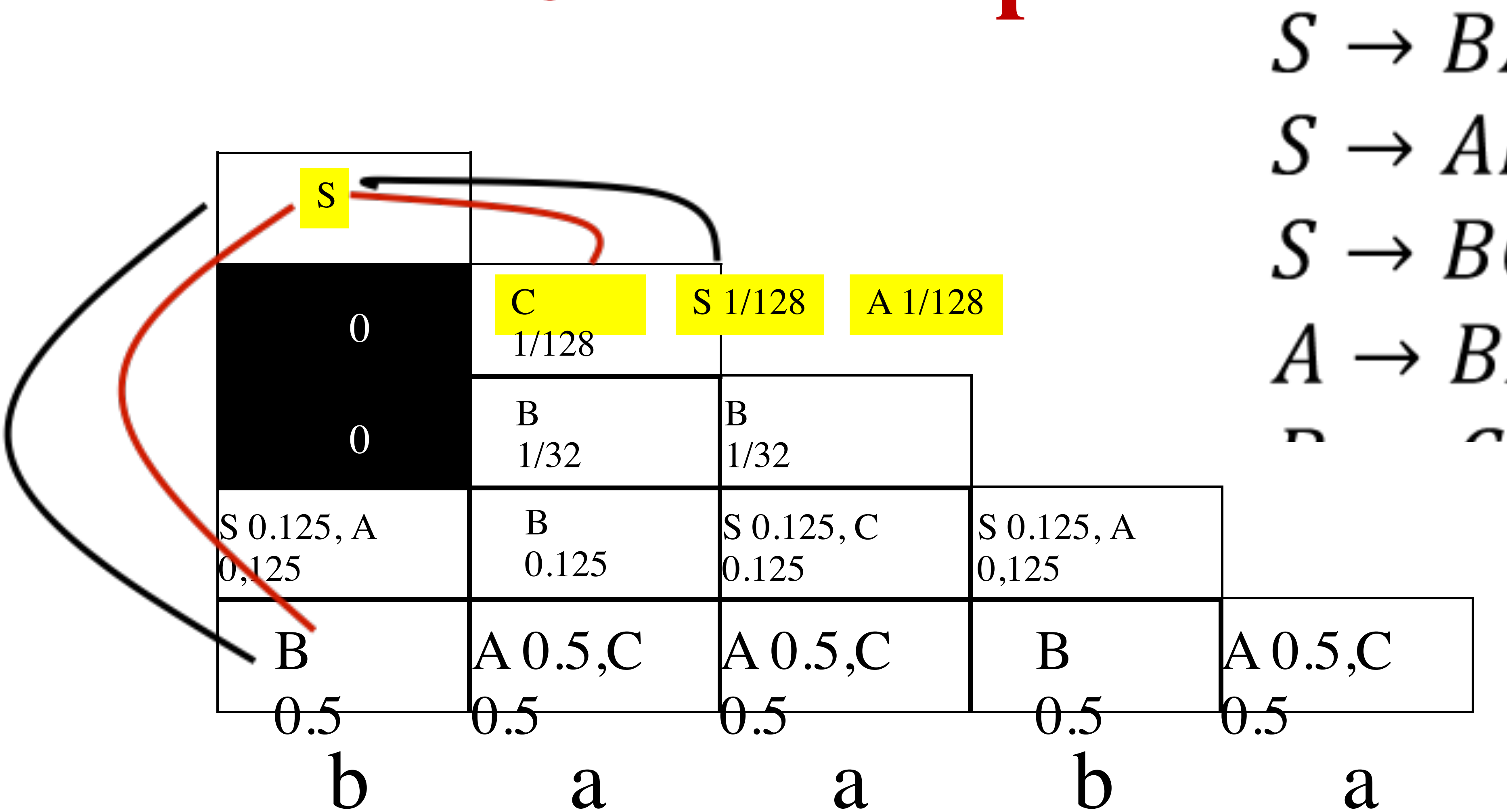
CYK example



$S \to B$
$S \to A$
$S \to B$
$A \to B$

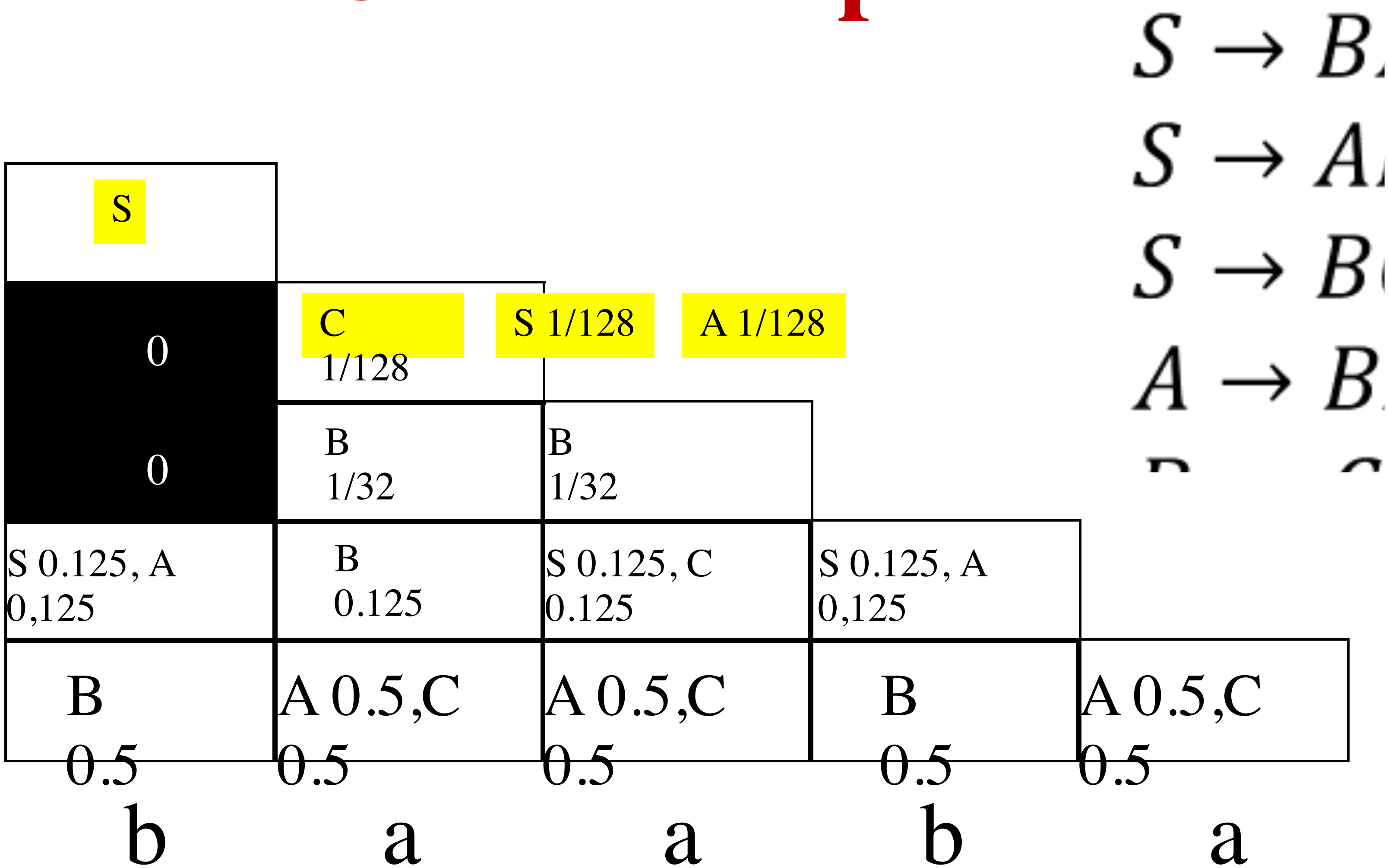
- When a rule has two or more possible productions, pick the most probable one

CYK example



- The same algorithm also applies to *weighted* PCFGs

CYK example



- What is the cost of parsing a string of N words with R rules?

CYK, PCFG and Structured prediction..

- The task we just performed..

- Assigning probabilities to entries from a very large set
  - Modelled a probability distribution over all possible parse trees
  - Selected the most probable parse
  - (How would you find the second most possible?)
- *Structured prediction*

Parsing CFGs

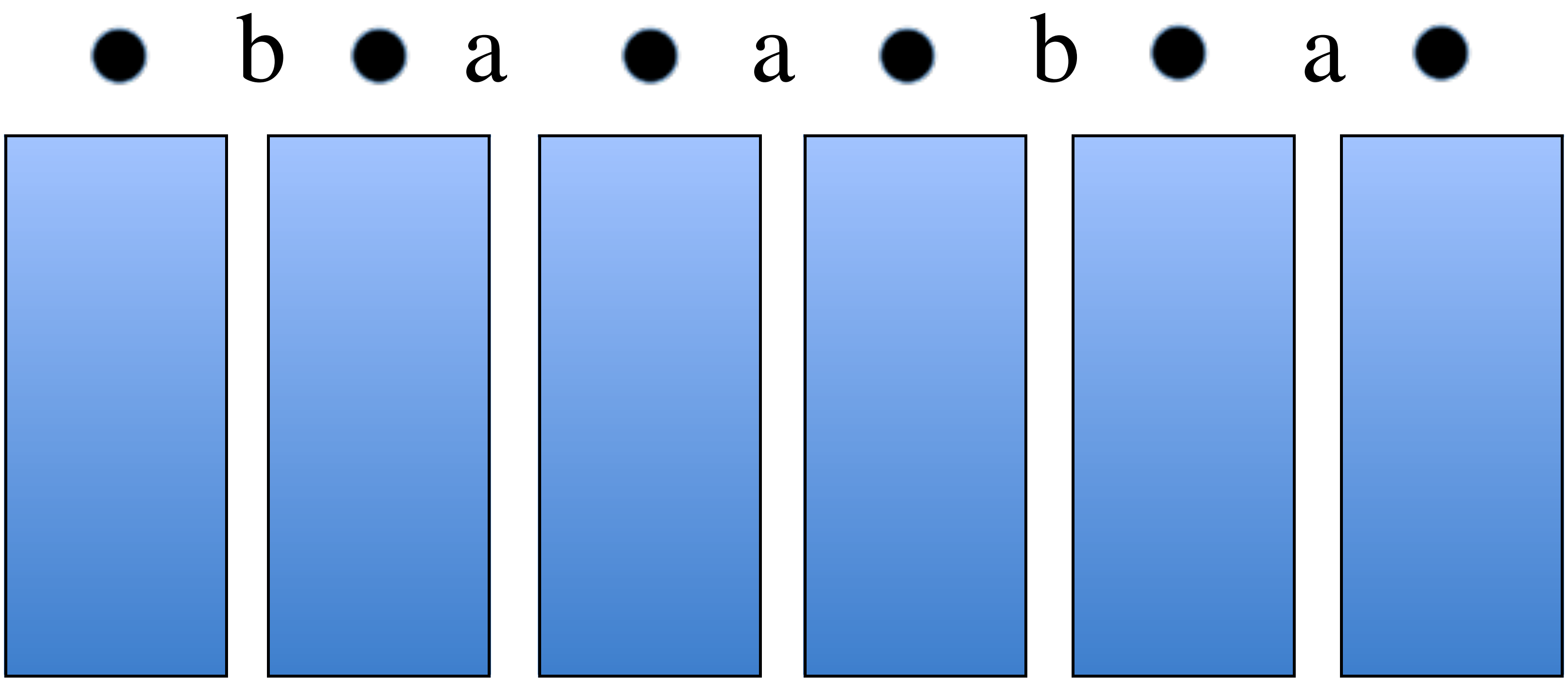
- The CYK parser is actually very expensive and inefficient
  - Nobody really uses it anymore except for very simple task
- A more efficient method is the *Earley* parser
  - Jay Earley, 1968, CMU
    - Then he gave it all up and became a shrink investigating his "inner critic"..
      - It's a very complicated looking algorithm

CYK vs Earley

- CYK: **Bottom up**
  - Build all possible* trees that can be built from the word sequence
  - Find which conforms to grammar
  - **Check conformance to grammar while building trees on words, to keep restrict computation*

- Earley: **Top down**
  - Build all possible* trees that can be produced by grammar

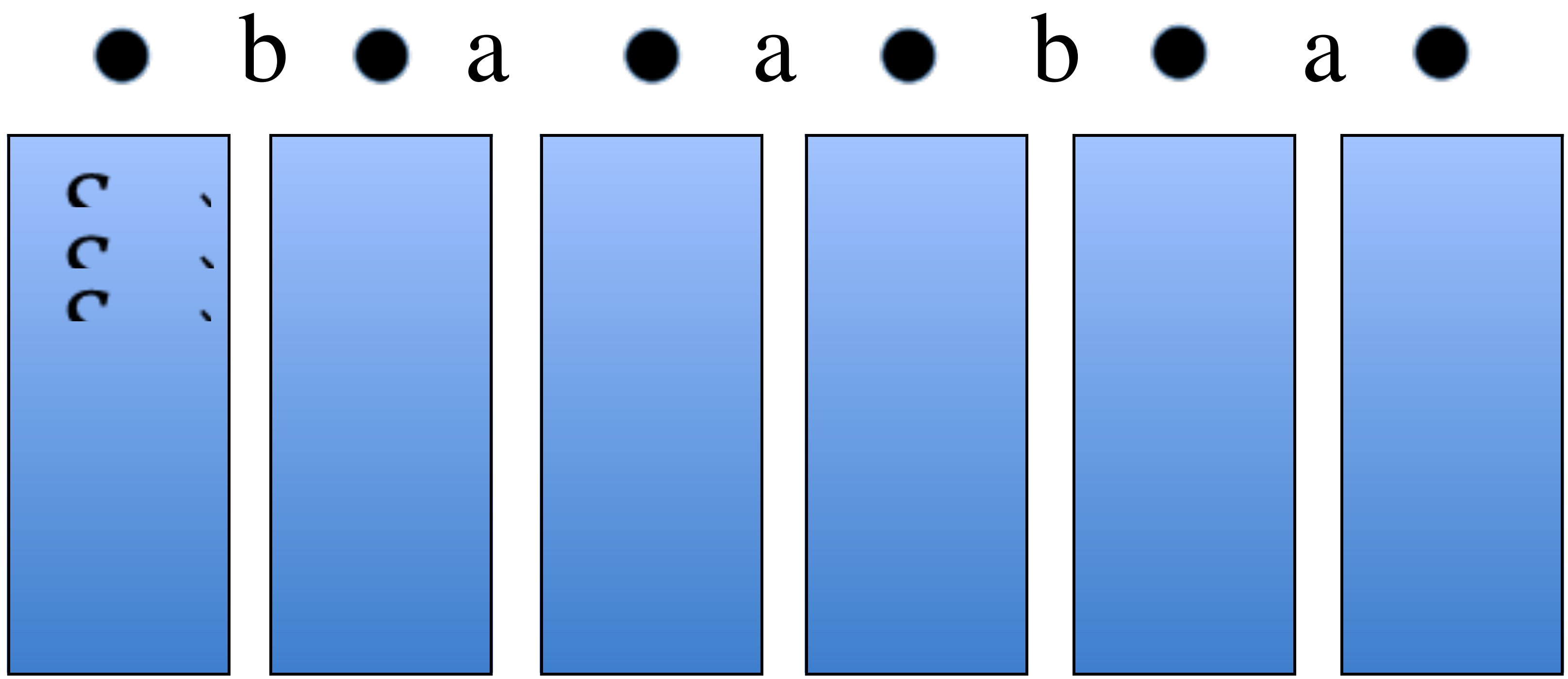
Earley example



$S -$
$S -$
$S -$
$A -$

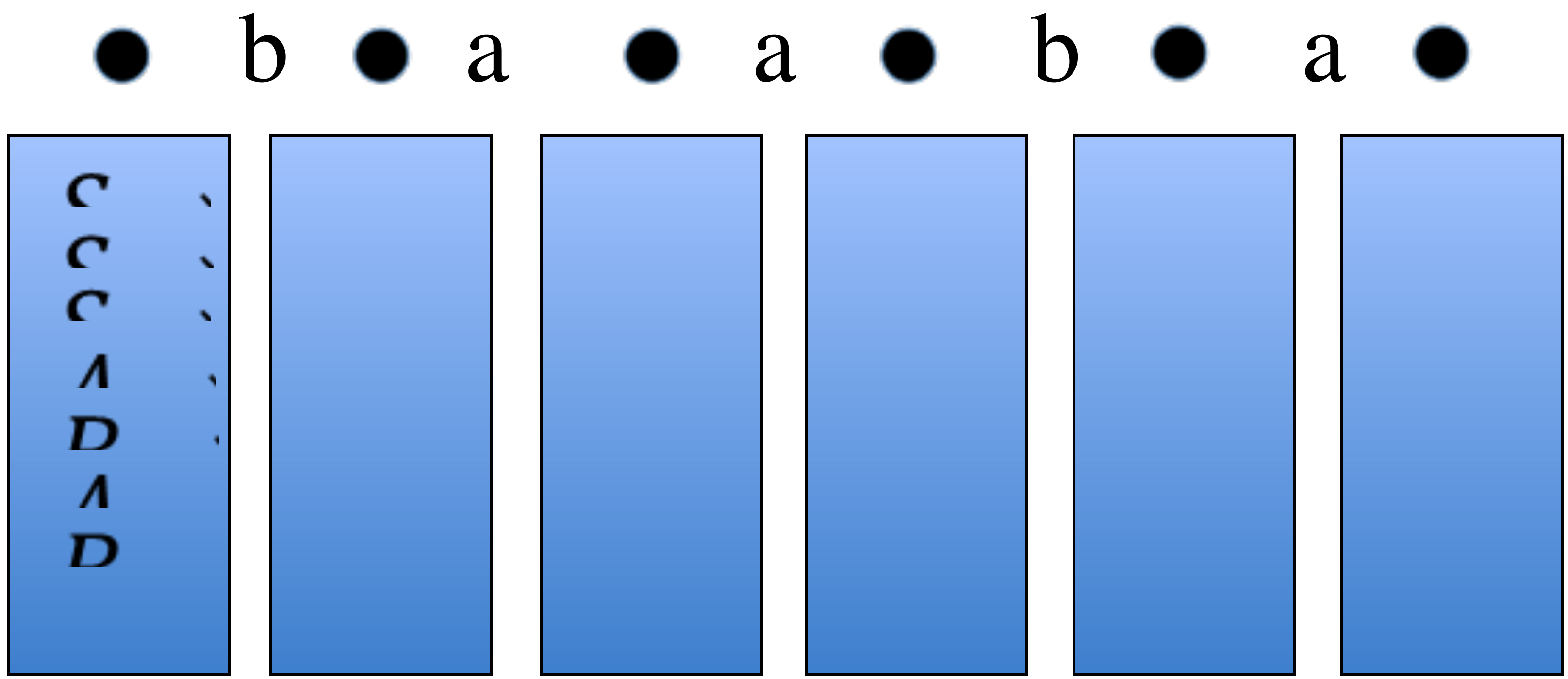
- We will maintain a list of partially expanded rules at each of the shown locations
  - We will go left to right

Earley example



- The star indicates the position until which we've successfully built a constituent

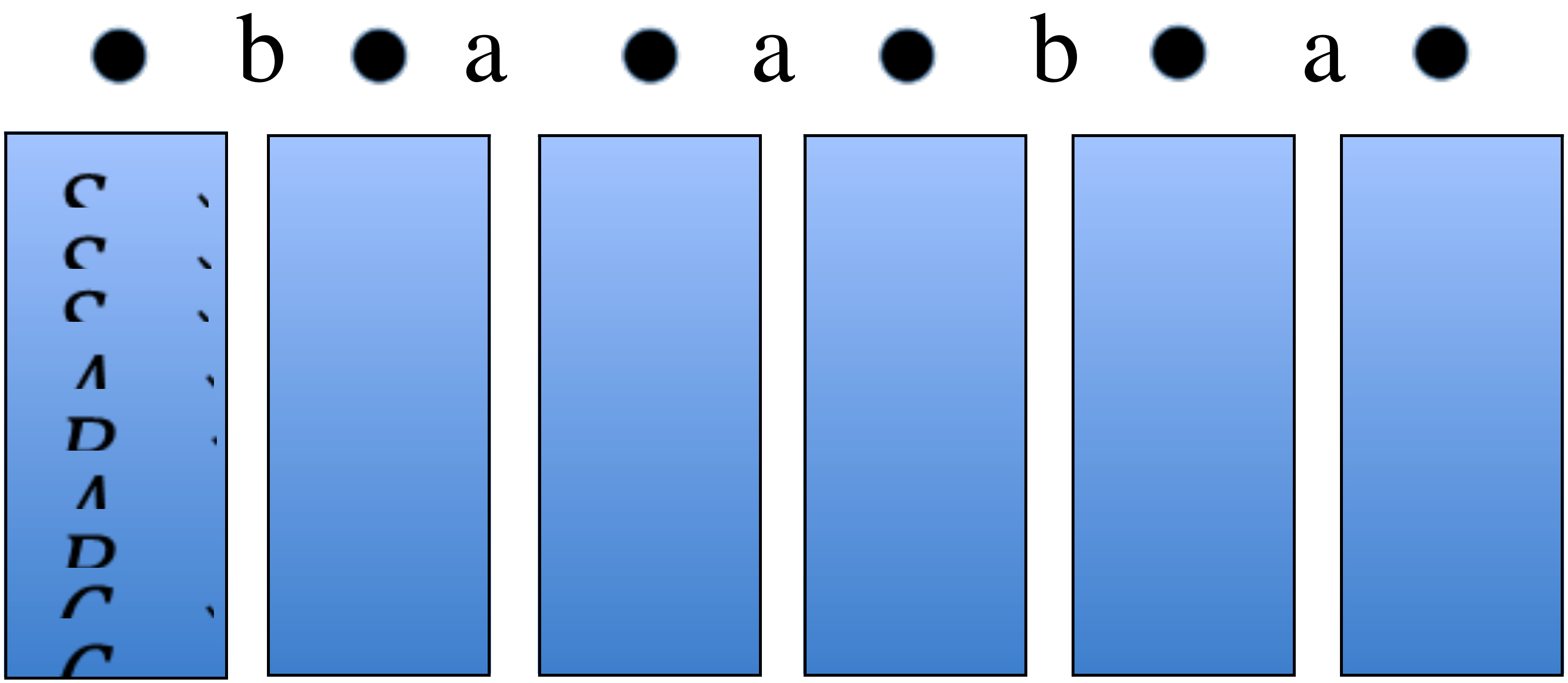
Earley example



“predict”

- The first symbols are A and B. Expand them
  - Include all possible expansions of A and B in one pass

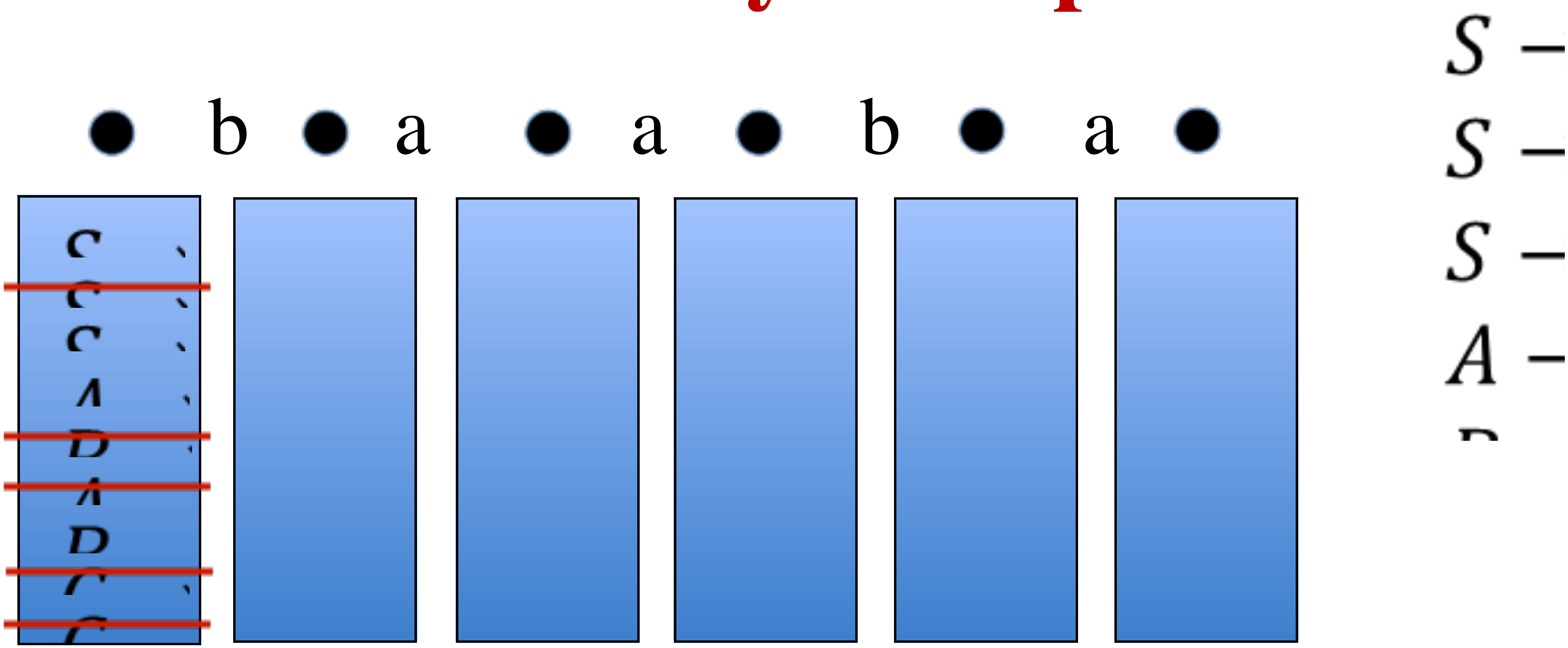
Earley example



"predict"

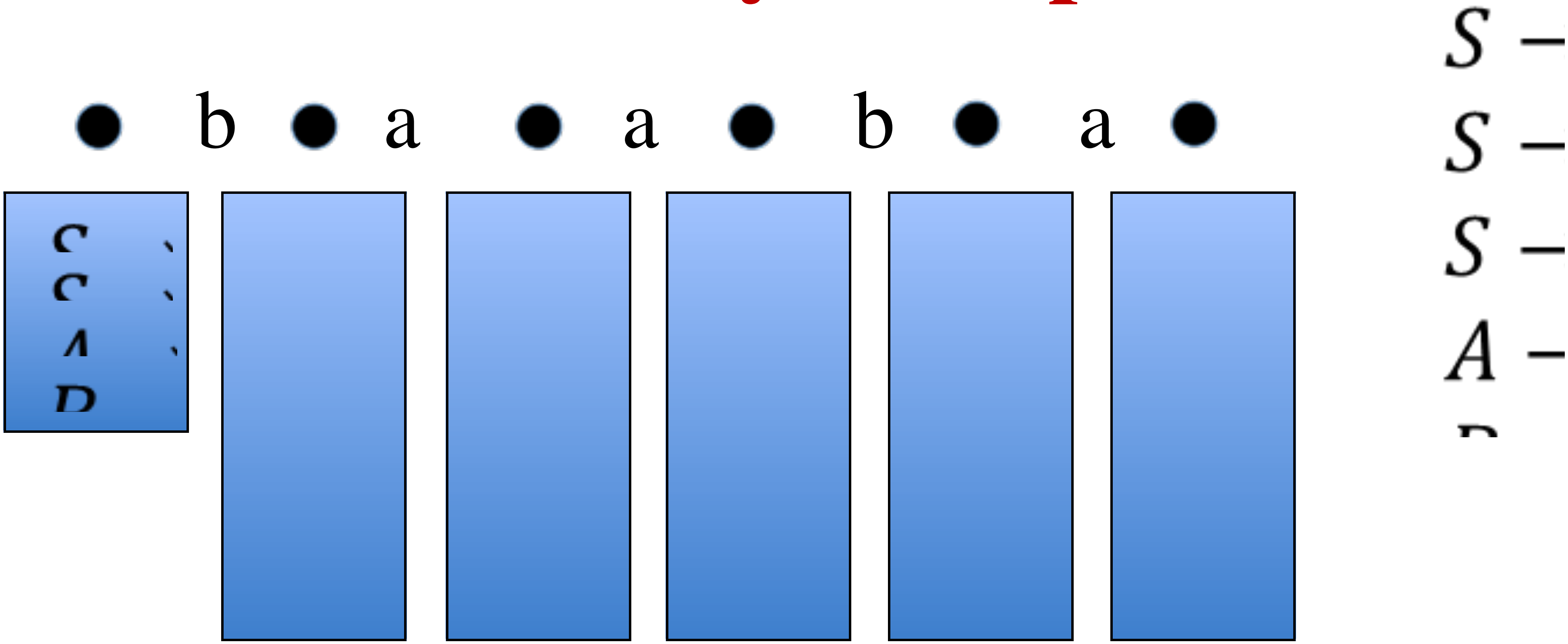
- We now have a new non-terminal C. Expand it
  - Don't revisit already-expanded NTs or you'll have an infinite loop

Earley example



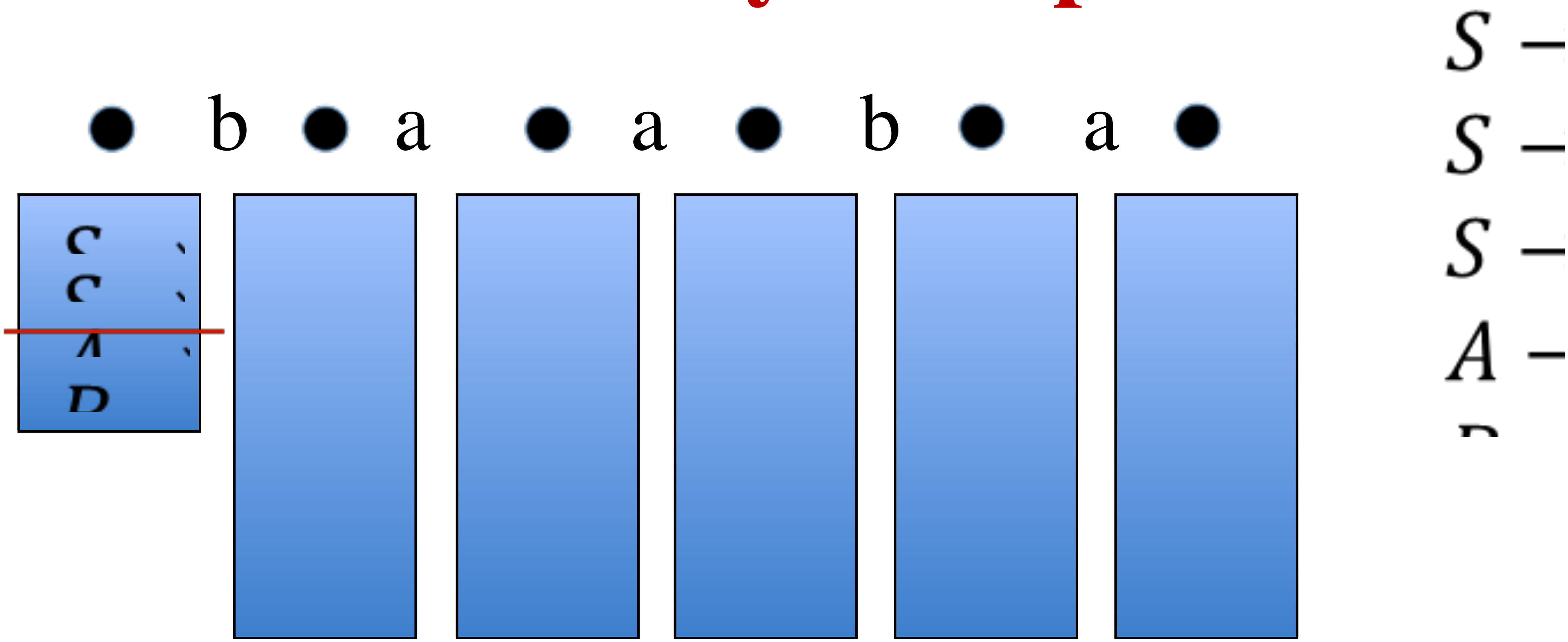
- Only one of the terminal-producing rules is valid for the upcoming symbol (b)
  - Retain it and its predecessors. Kill all the rest

Earley example



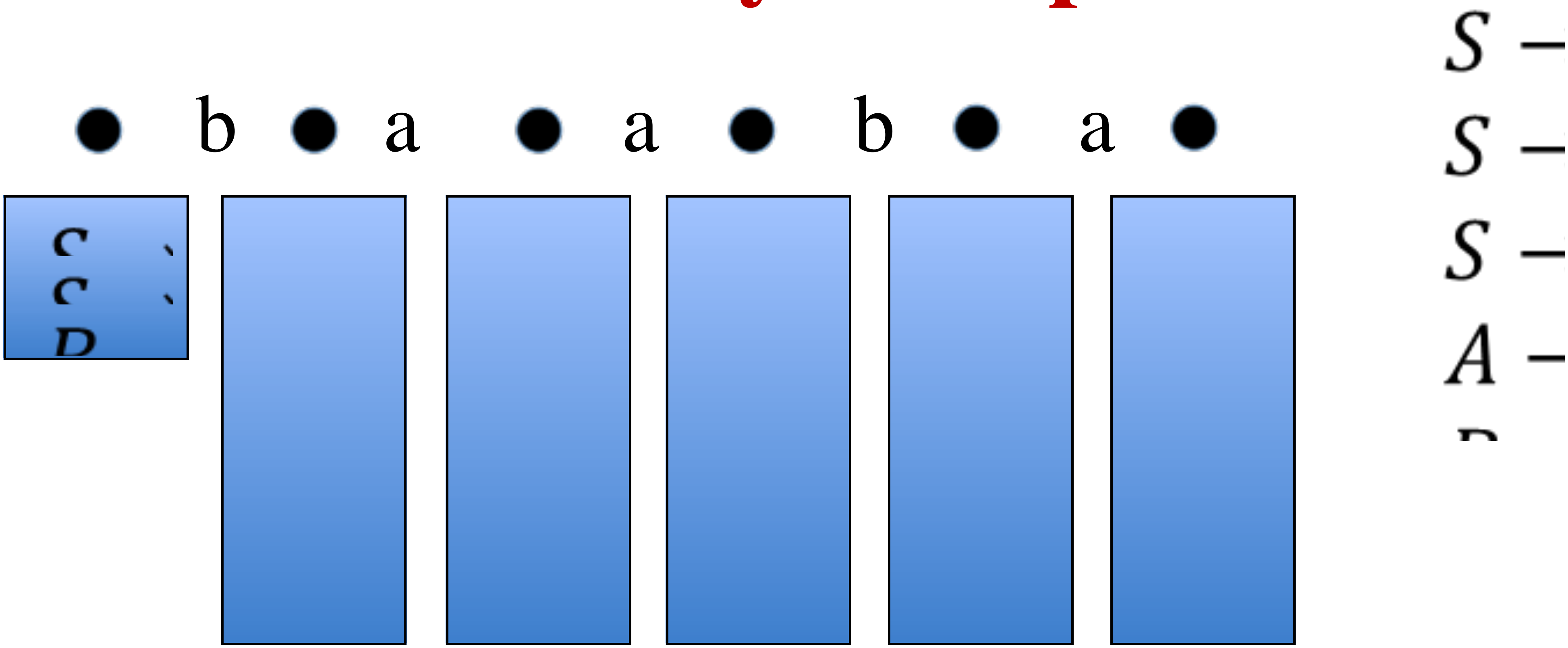
- Only one of the terminal-producing rules is valid for the upcoming symbol (b)
  - Retain it and its parents. Kill all the rest

Earley example



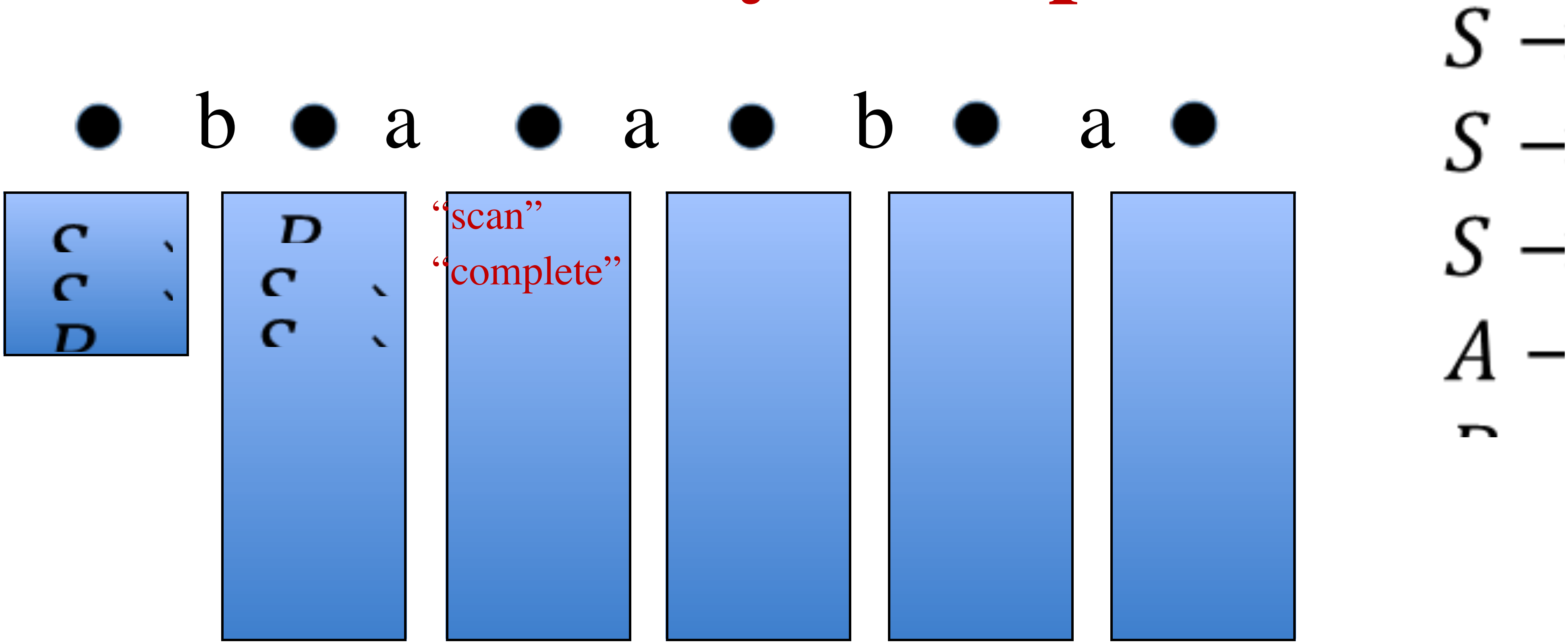
- Can also kill orphaned rules that don't lead to "S"

Earley example



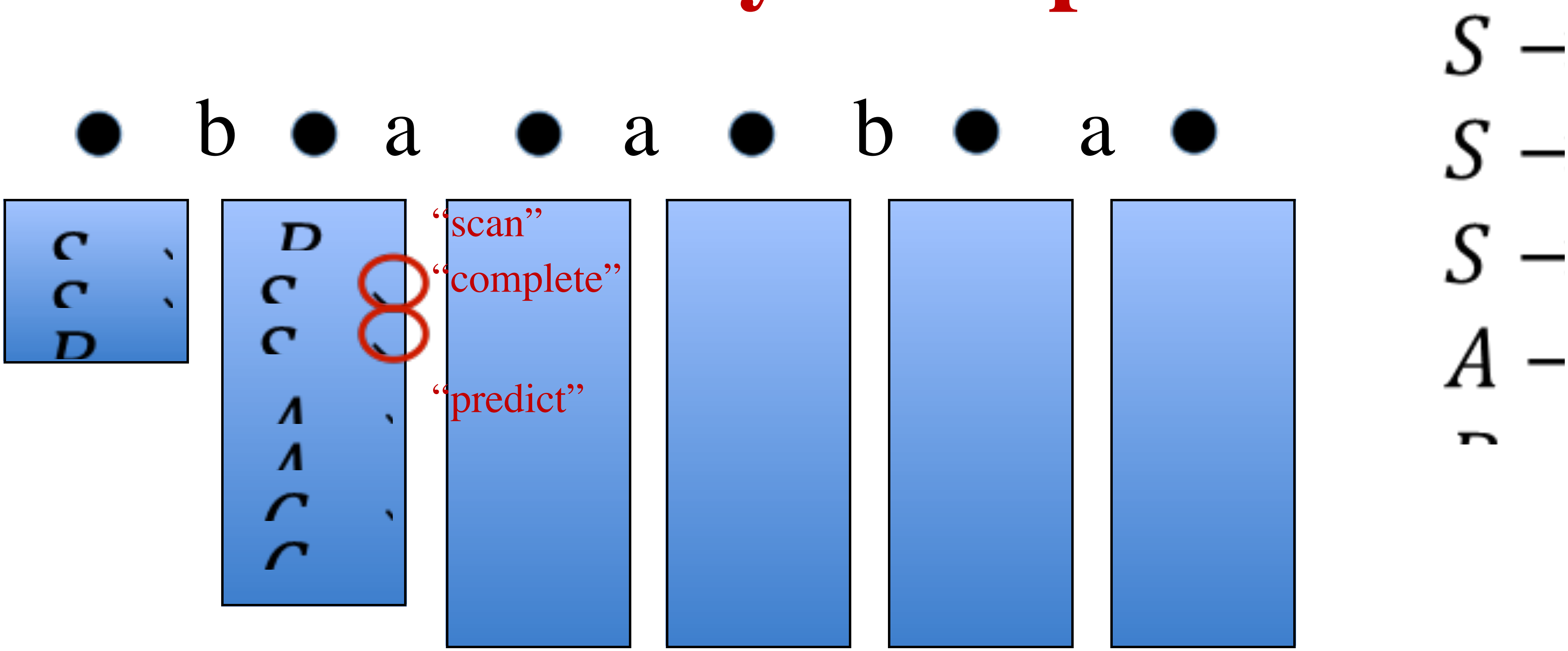
- Can also kill orphaned rules that don’t lead to “S”

Earley example



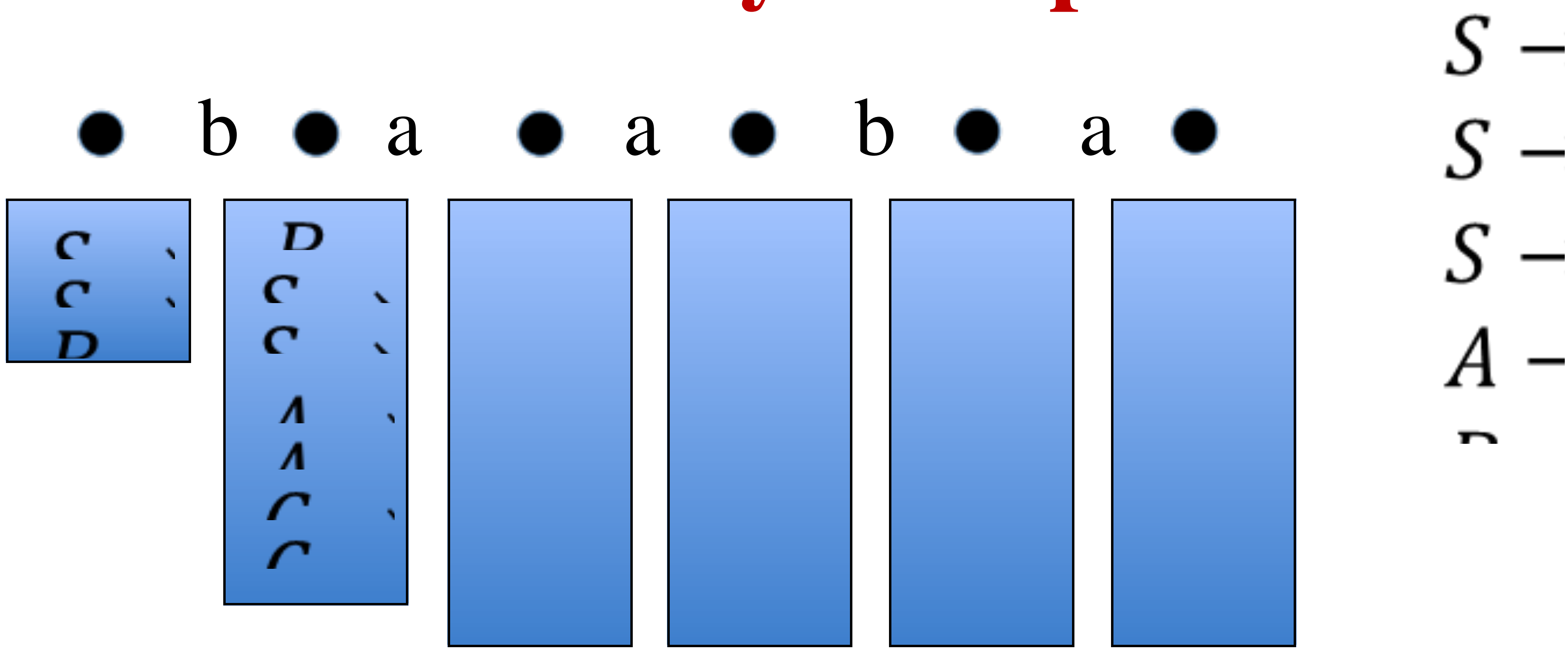
- "Scan" the "b": Move the terminal rules that produce b over to the second column
  - Move the star to show "b" is consumed

Earley example



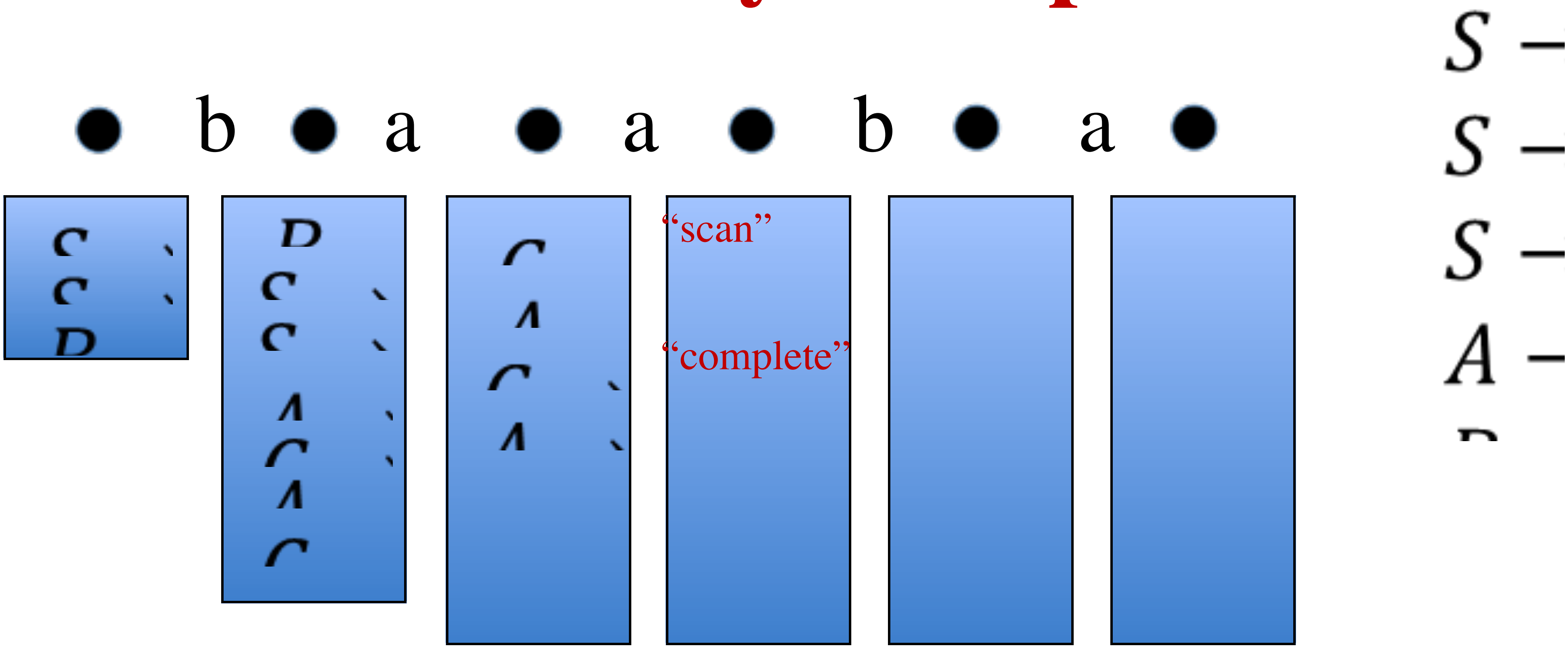
- Recursively expand the first NT in each of the rules until we get to terminals
  - "Scan" their expansions to determine if any of them

Earley example



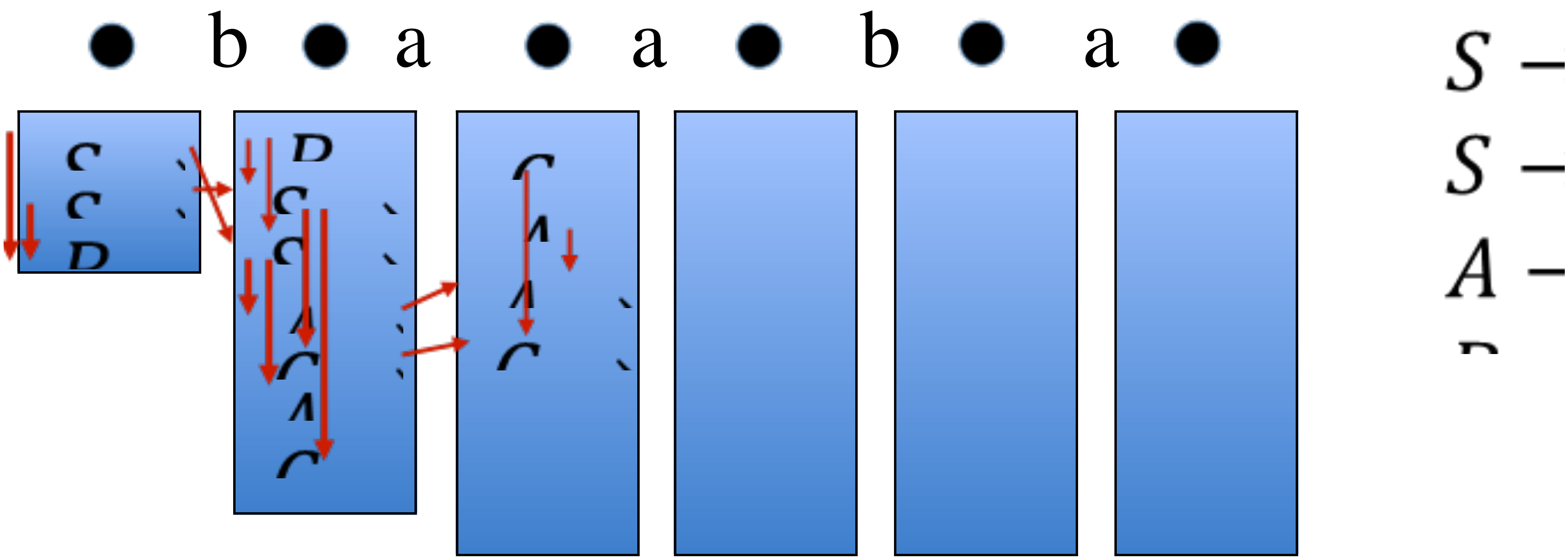
- Recursively expand the first NT in each of the rules until we get to terminals
  - "Scan" their expansions to determine if any of them

Earley example



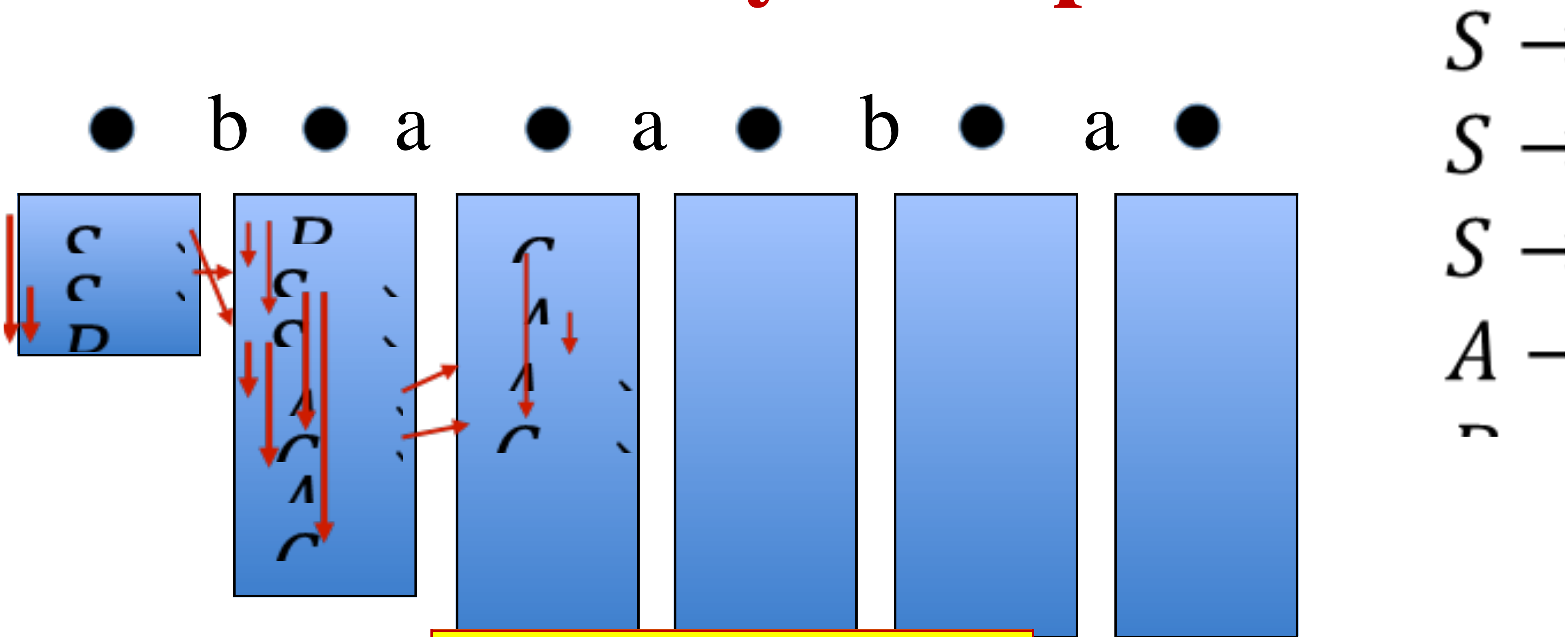
- Move the rules that can produce the “a” and their parents to the next column
  - Move the *

Earley example



- For all rules, keep track of parents
- For a valid string, we will get at least one
- If we get multiple such expansions we h

Earley example



How to disambiguate?

- For all rules, keep track of parents
- For a valid string, we will get at least one
- If we get multiple such expansions we h

Earley with PCFGs

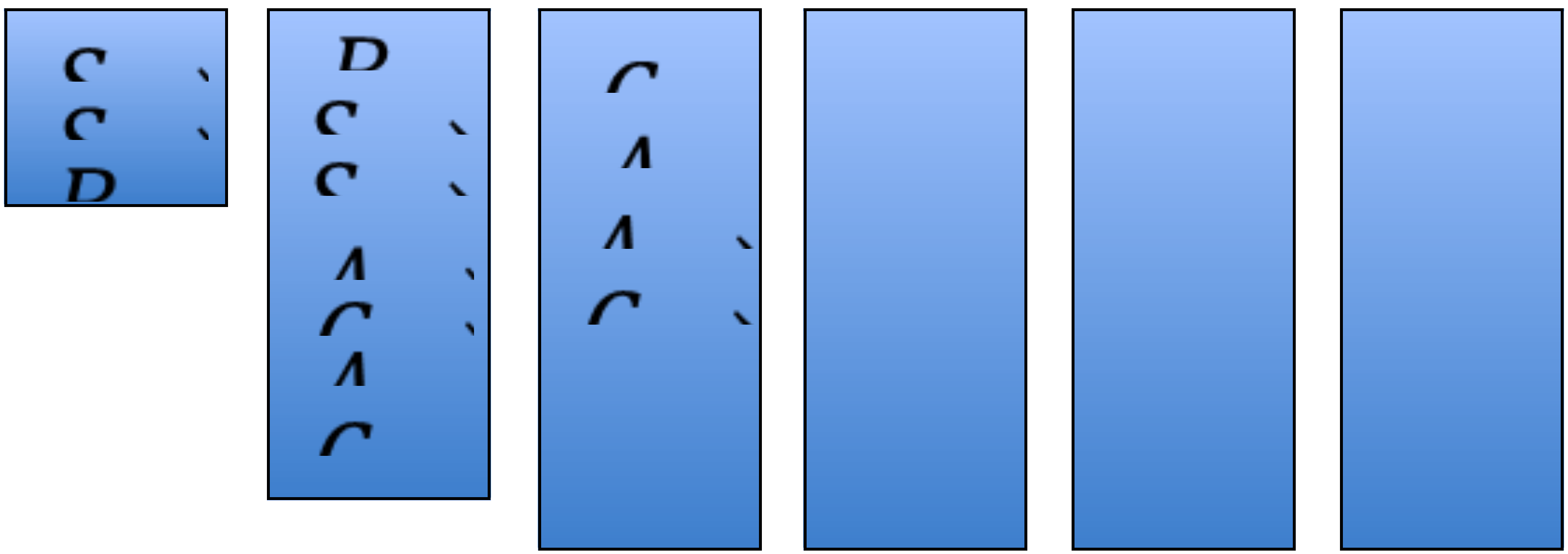
$S \to B$

$S \to A$

$S \to B$

$A \to B$

• b • a • a • b • a •



- How?

Evaluation

- Take a sentence from the test set.
- Use your parser to propose a hypothesis parse.
- Treebank gives you the correct parse.
- Precision and recall on labeled (or unlabeled) constituents.
  - Also, average number of crossing brackets (compared to correct parses) in your hypotheses.
- The training/development/test split has been held constant for a long time; possibly a cause for concern.

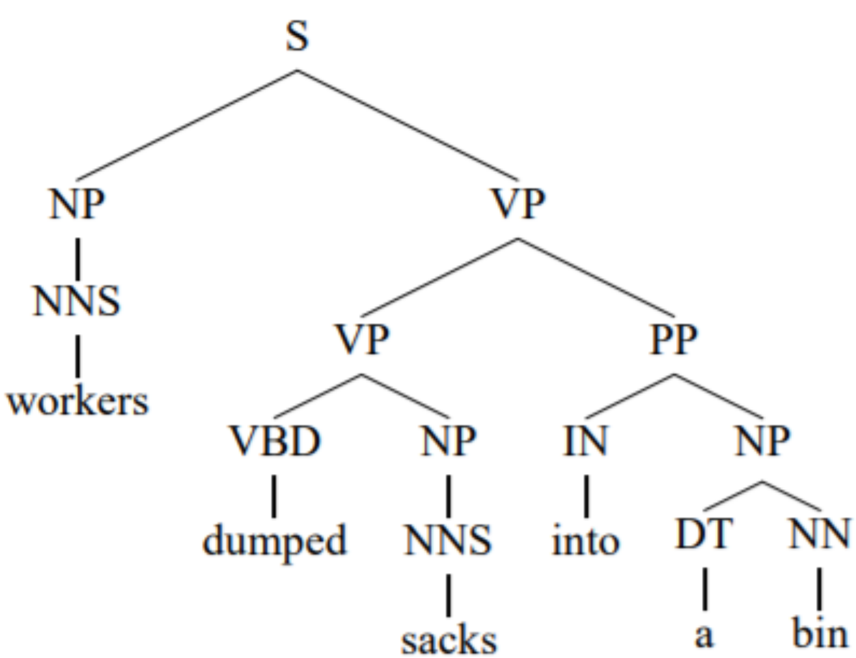
Parsing in Reality

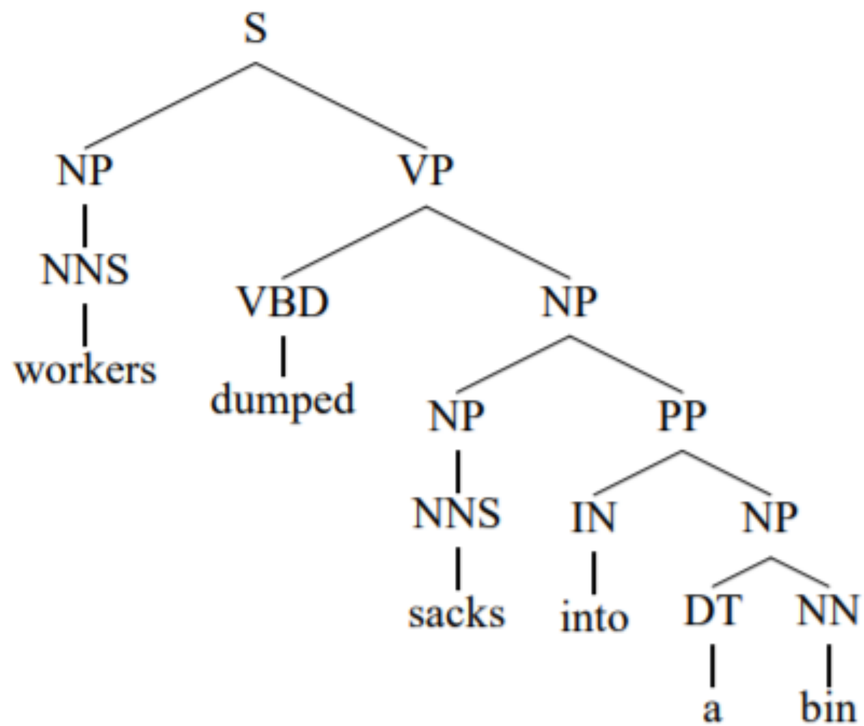
- Generally speaking, few industrial-strength parsers actually call CKY or Earley's.
- Extensions to the basic CFG model (next topic) make reduction to CFG expensive.
- Standard techniques:
  - Beam search
  - Agenda-based approximations with pruning and/or A*
  - "Coarse-to-fine"

The problem

- The basic parsers are inefficient
- The PCFG structure enables us to disambiguate, but are nevertheless insufficient to actually model the language

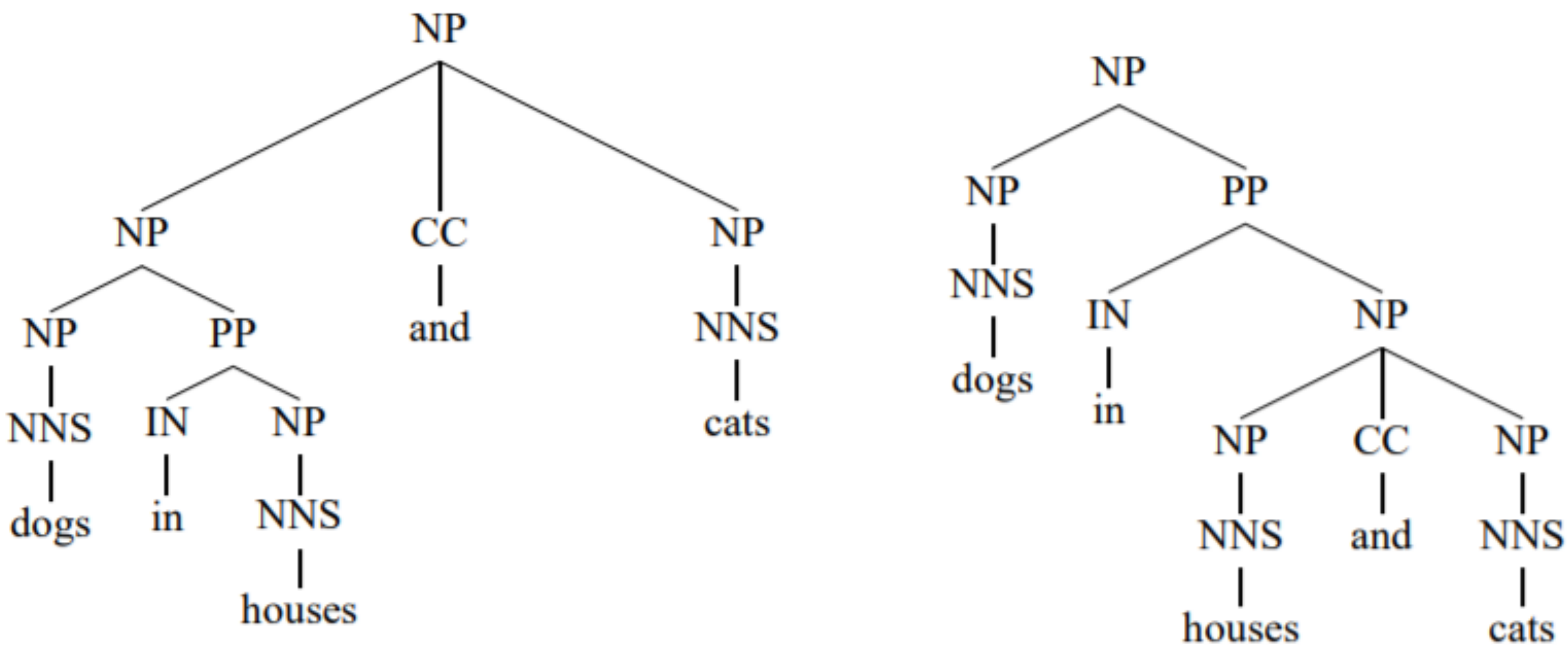
Examples of ambiguous parses





- From Michael Collins
  - Prepositional attachment ambiguity

Examples of ambiguous parses

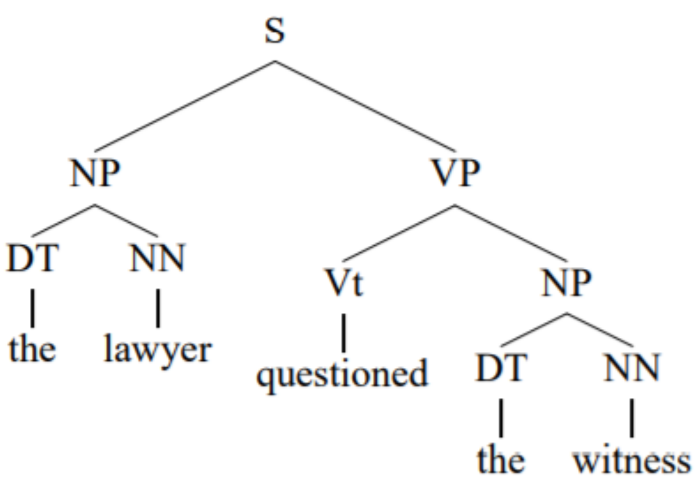


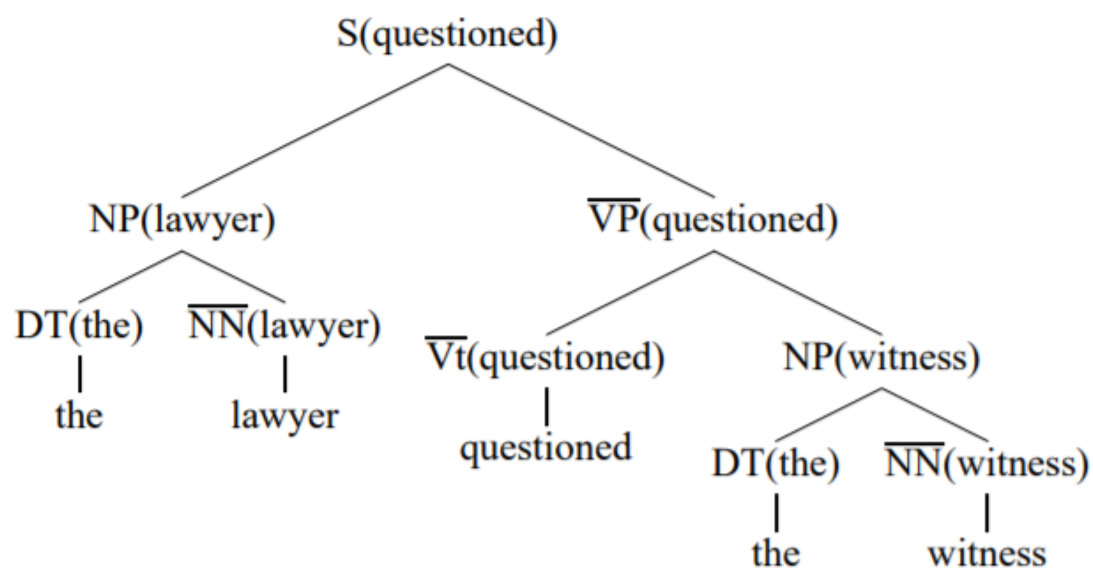
- From Michael Collins
  - Coordination ambiguity: Identical set of rules applied, only difference is the order

Solution: Lexicalization

-
  - Attach to each non-termina
    it can construct
    - $S \rightarrow NP\, VP$
    - $S(questioned) \rightarrow NP(law$
  - Each rule now multiplies

Lexicalized Parse



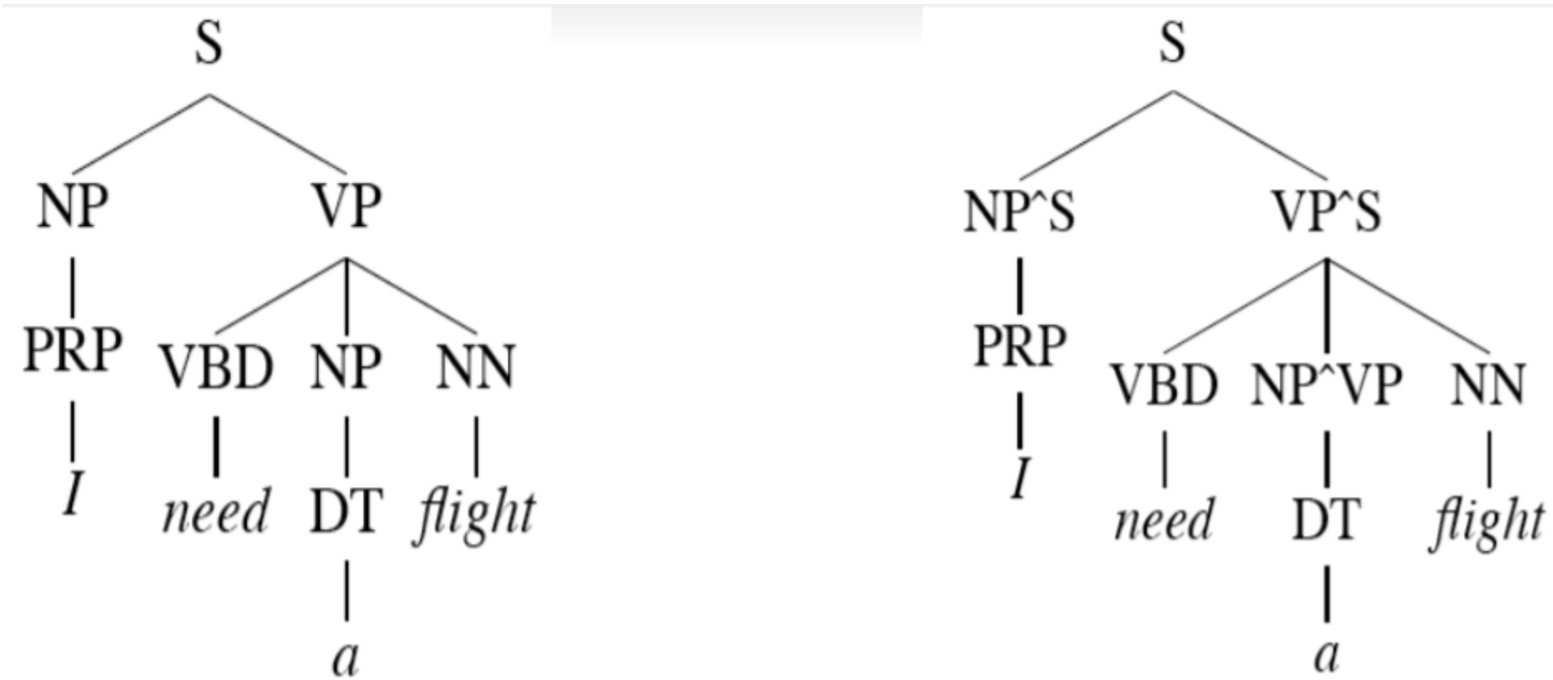


Natural language is not context independent..

- But CFGs are easy to handle
- Compromise?

- Context *free* grammar
  - But *parent-dependent* probabilities
  - Like an expanded markov state in Markov chains

Parent annotation

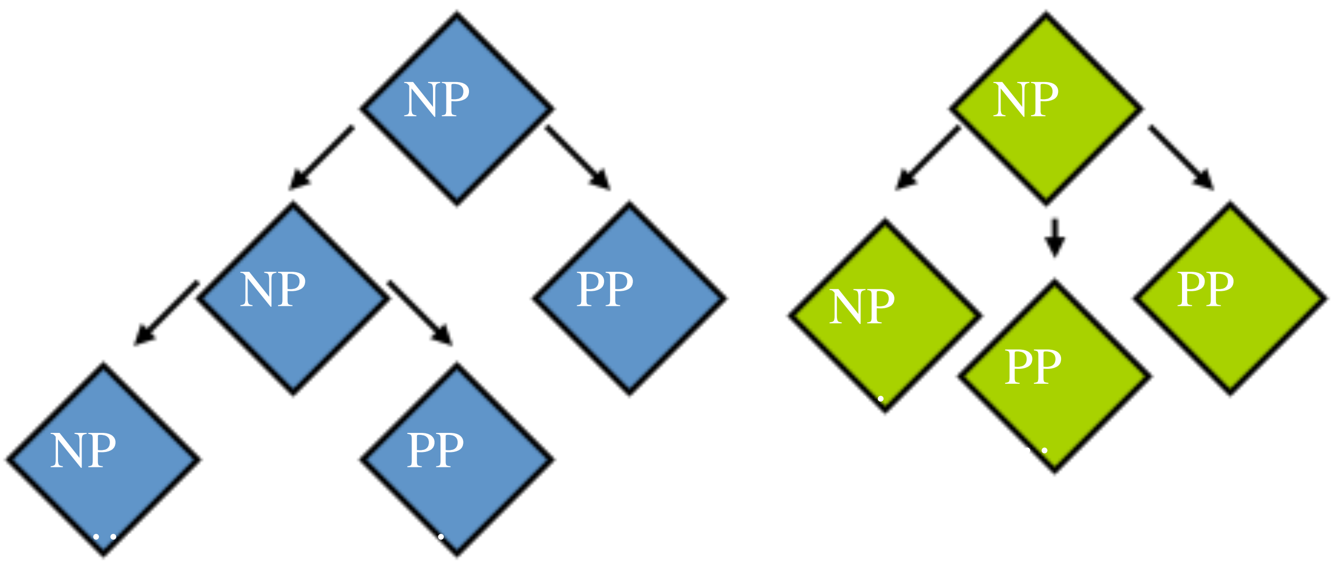


- Rules are cloned for different parents
  - Parent-specific probability distributions over expansions
  - But the actual rules remain basic CFG rules!

Parent Annotation

NP → *p* NP

NP → *q* NP PP

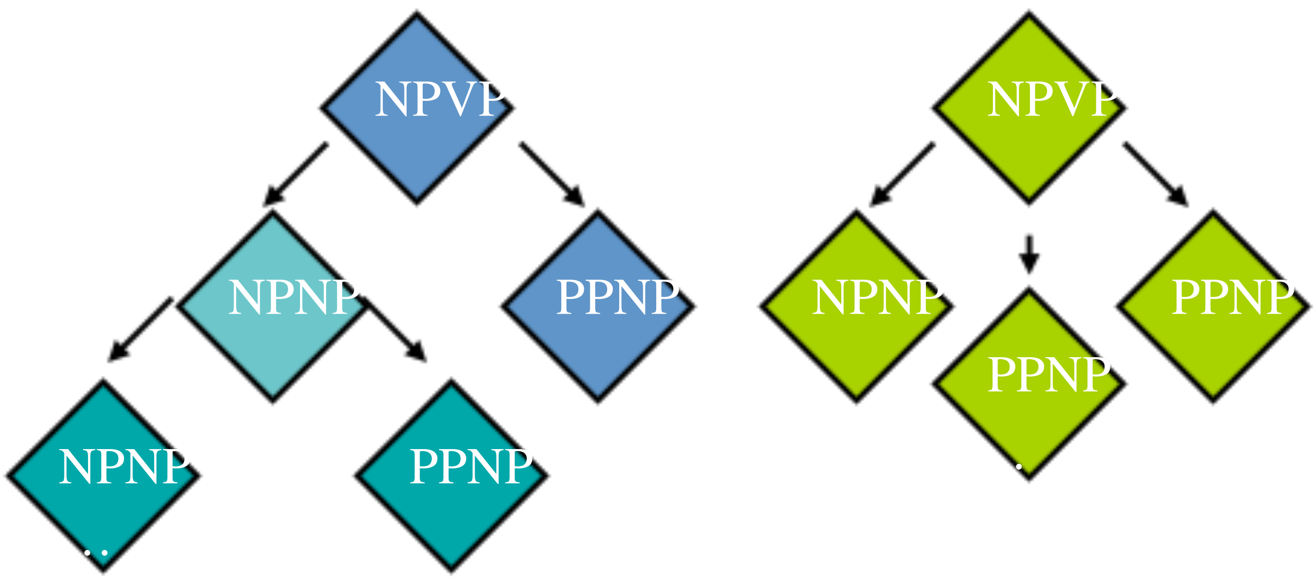


Parent Annotation

NPVP $\rightarrow_p$ NPNP

NPNP $\rightarrow_r$ NPNP

NPVP $\rightarrow_q$ NPNP



Parent Annotation

- Another way to think about it …
- Before: $$p(\text{tree}) = \prod_{n \in nodes(\text{tree})} \rho(childsequence(n) \mid n)$$
- Now: $$p(\text{tree}) = \prod_{n \in nodes(\text{tree})} \rho(childsequence(n) \mid n, parent(n))$$
- This could conceivably **help** performance (weaker independence assumptions)
- This could conceivably **hurt** performance (data sparseness)

Parent Annotation

- From Johnson (1998):
- PCFG from WSJ Treebank: 14,962 rules
- Of those, 1,327 would **always** be subsumed!
- After parent annotation: 22,773 rules
- Recall 69.7% -> 79.2%; precision 73.5% -> 80.0%

Head Annotation

- "I love all my children, but one of them is **special**."

S –> NP

VP –> VBD

NP –> DT NNS

- Heads not in the Treebank.
- Usually people use **deterministic head rules** (Magerman, 1995).

Algorithms

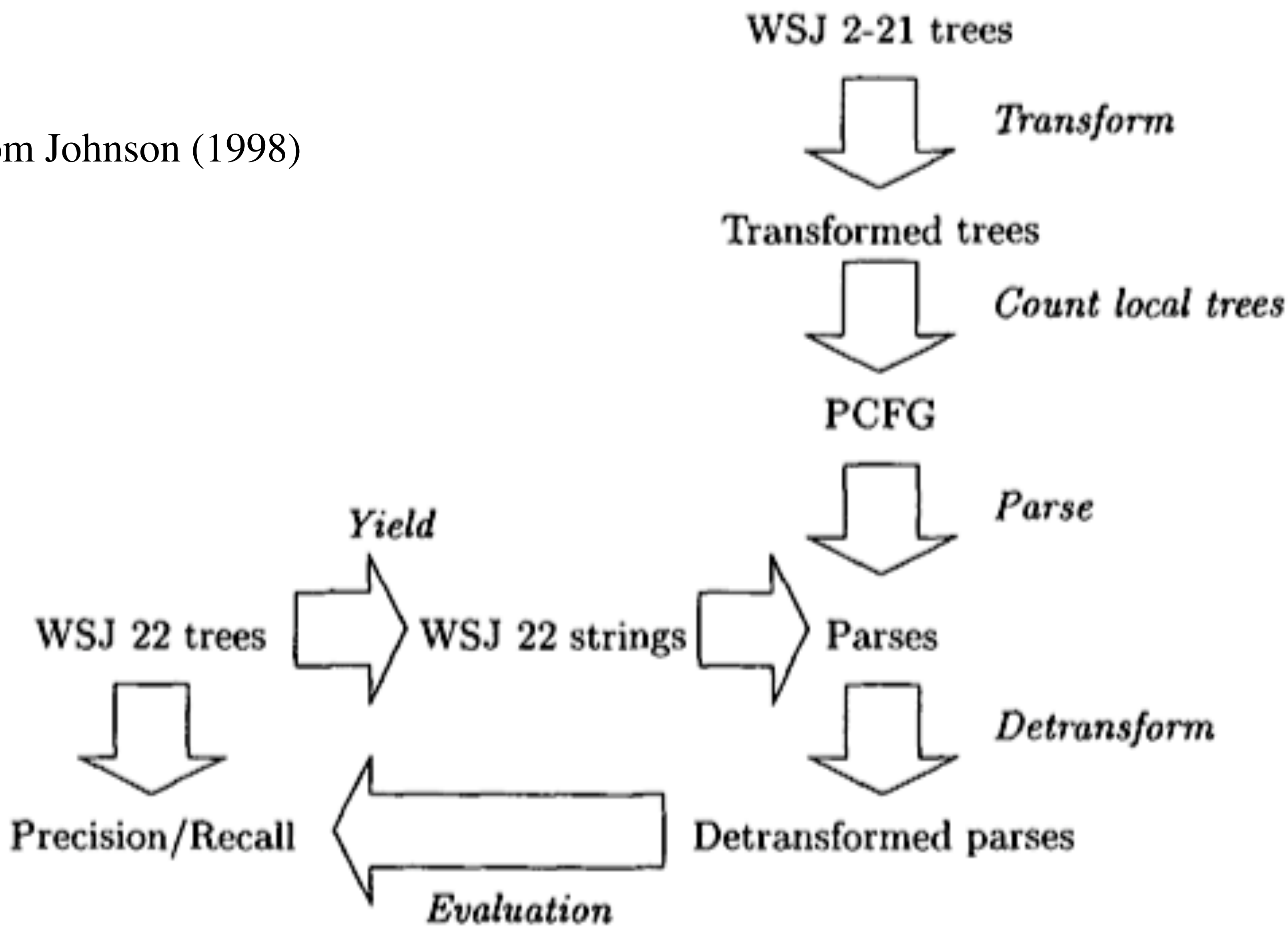
- These “decorations” affect our parser’s runtime.
  - Why?
  - Any ideas about how to get around this?

Some Famous Parsers

Training Parsers In Practice

- Transformations on trees
  - Some of these are generally taken to be crucial
  - Some are widely debated
  - Lately, people have started **learning** these transformations
- Smoothing is crucial; the grammars that result from transformed trees have lots more rules and therefore more parameters.

from Johnson (1998)



Collins Model 1 (1997)

- Trees are headed and lexicalized
  - What's the difference?
- Huge number of rules!

VPsaw → Vsaw NPman PPthrough

VPsaw → Vsaw NPman PPwith

VPsaw → Vsaw NPwoman PPthrough

VPsaw → Vsaw NPman

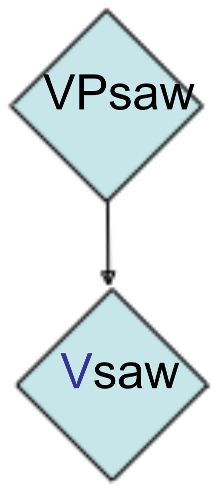
- Key: factor probabilities within rule.

Collins Model 1 (1997)

- Everything factors down to rules, then further. We're given the parent nonterminal and head word.

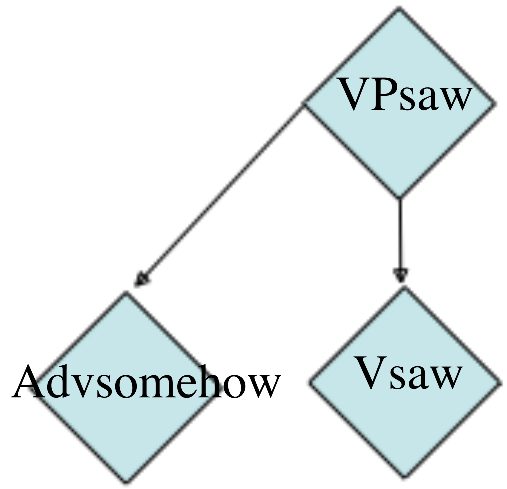
Collins Model 1 (1997)

- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.



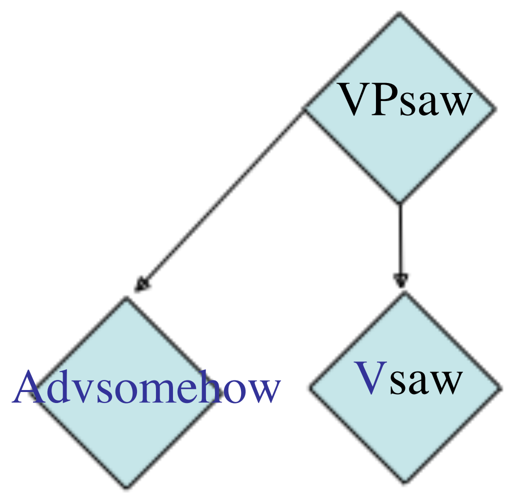
Collins Model 1 (1997)

- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.
- Generate a sequence of left children.



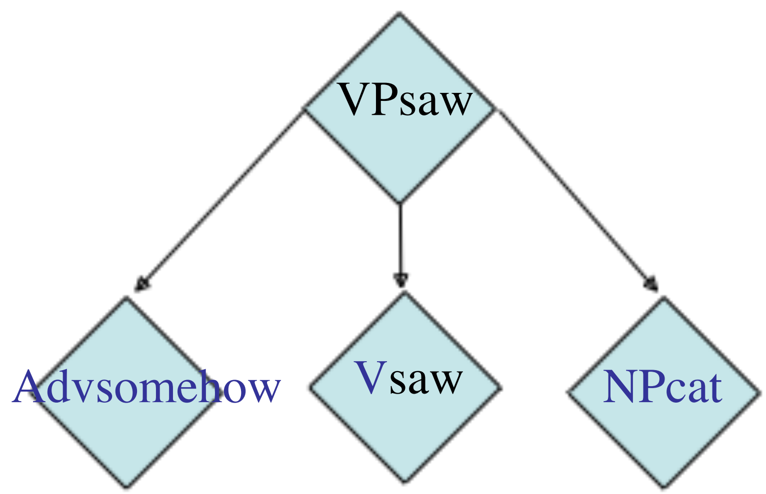
Collins Model 1 (1997)

- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.
- Generate a sequence of left children.



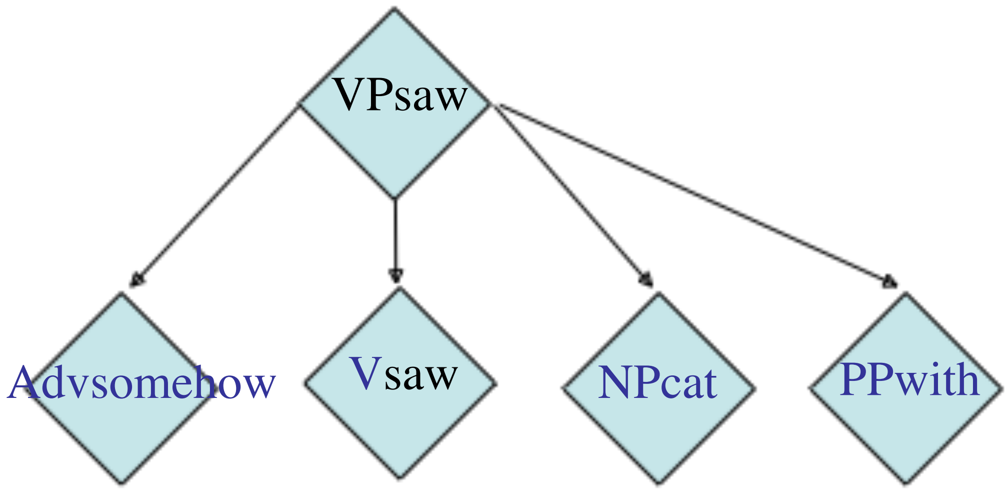
Collins Model 1 (1997)

- Everything factors down to rules, then further. We’re given the parent nonterminal and head word.
- Randomly generate the head child’s nonterminal.
- Generate a sequence of left children.
- Then right.



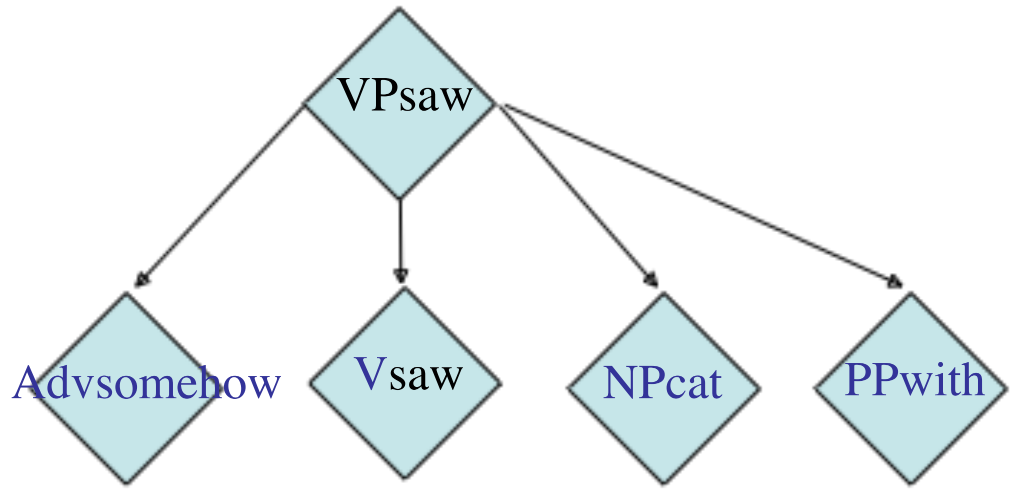
Collins Model 1 (1997)

- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.
- Generate a sequence of left children.
- Then right.



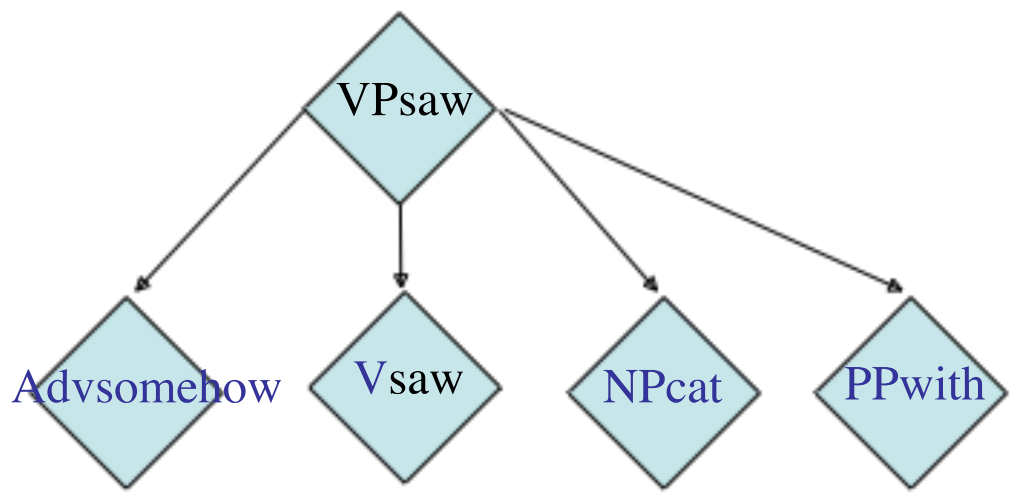
Collins Model 1 (1997)

- Everything factors down to rules, then further. We're given the parent nonterminal and head word.
- Randomly generate the head child's nonterminal.
- Generate a sequence of left children.
- Then right.



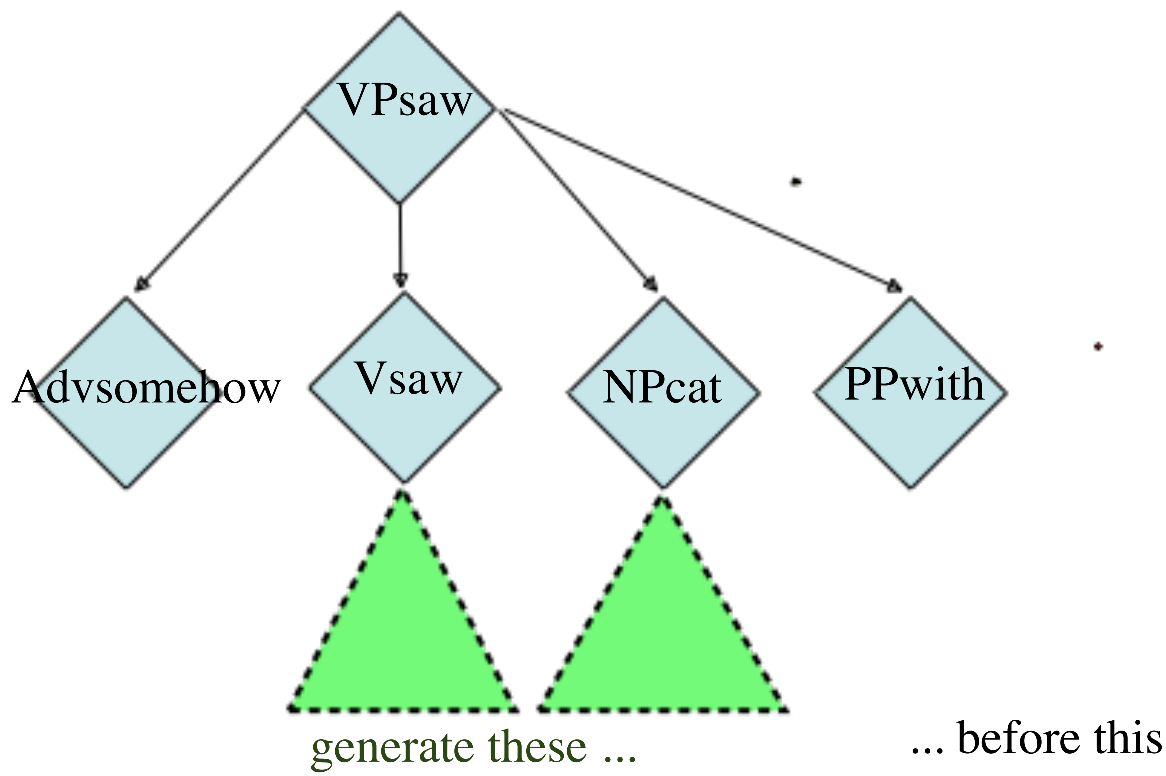
Collins Model 1 (1997)

- Interesting twist: want to model the **distance** between head constituent and child constituent. How?



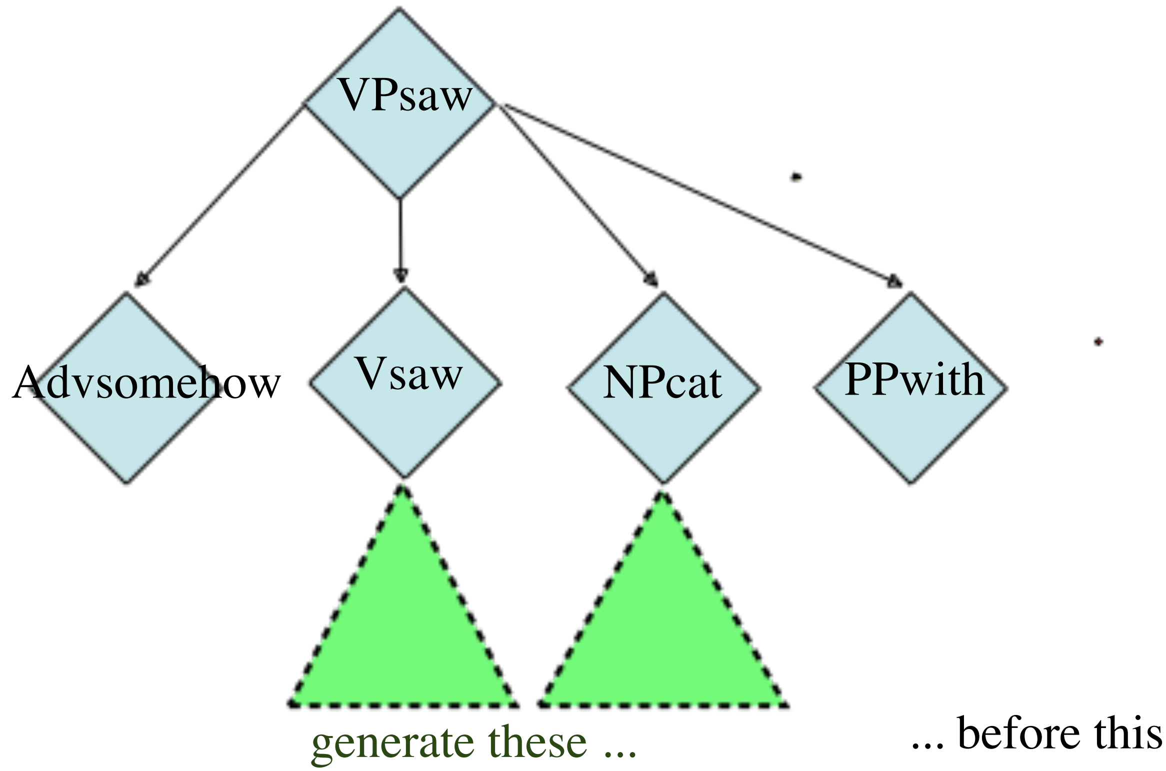
Collins Model 1 (1997)

- Interesting twist: want to model the **distance** between head constituent and child constituent. How?
- Depth-first recursion.



Collins Model 1 (1997)

- Interesting twist: want to model the **distance** between head constituent and child constituent. How?
- Depth-first recursion.
- Condition next child on features of the parent's yield so far.



Collins Model 1 (1997)

- Interesting twist: want to model the **distance** between head constituent and child constituent. How?
- Depth-first recursion.
- Condition next child on features of the parent's yield so far.

$$p(\mathrm{PP_{with}} \mid \mathrm{VP_{saw}}, \mathrm{right}, \text{“the cat who liked milk”}) \approx p(\mathrm{PP_{with}} \mid \mathrm{VP_{saw}}, \mathrm{right}, \mathrm{length} > 0, +\mathrm{verb})$$

$$\begin{aligned}
&p(L_n, u_n, L_{n-1}, u_{n-1}, \ldots, L_1, u_1, H, w, R_1, v_1, R_2, v_2, \ldots, R_m, v_m \mid P, w) \\
&\quad = \quad p(H \mid P, w) \\
&\qquad\quad \cdot \prod_{i=1}^{n} p(L_i, u_i \mid P, w, H, \mathrm{left}, \Delta_i) \\
&\qquad\quad \cdot p(\mathrm{stop} \mid P, w, H, \mathrm{left}, \Delta_{n+1}) \\
&\qquad\quad \cdot \prod_{i=1}^{m} p(R_i, v_i \mid P, w, H, \mathrm{right}, \Delta'_i) \\
&\qquad\quad \cdot p(\mathrm{stop} \mid P, w, H, \mathrm{right}, \Delta'_{n+1})
\end{aligned}$$

Collins Models 2 and 3 (1997)

- Model 2: Complements, adjuncts and subcategorization frames
  - Treebank decoration: -C on specifiers and arguments
  - Probability model: first pick set of complements (side-wise), must ensure they are all generated
  - *the issue was a bill funding Congress*
- Model 3: Wh-movement and extraction
  - Treebank decoration: “gap feature”
  - Probability model: gap feature “passed around the tree,” must be “discharged” as a trace element.
  - *the store that IBM bought last week*

Other Points

- Unknown words at test time: any training word with count < 6 becomes UNK
- Smoothing: deleted interpolation
- Tagging is just part of parsing (not a separate stage)
- Markov order increased in special cases:
  - within base noun phrases (NPBs) - first order
  - conjunctions ("and") predicted together with second conjunct
  - punctuation (details in 2003 paper)

Practical Notes

- Collins parser is freely available
- Dan Bikel replicated the Collins parser cleanly in Java
  - Easier to re-train
  - Easier to plug-and-play with different options
  - Multilingual support
  - May be faster (with current Java) - I'm not sure

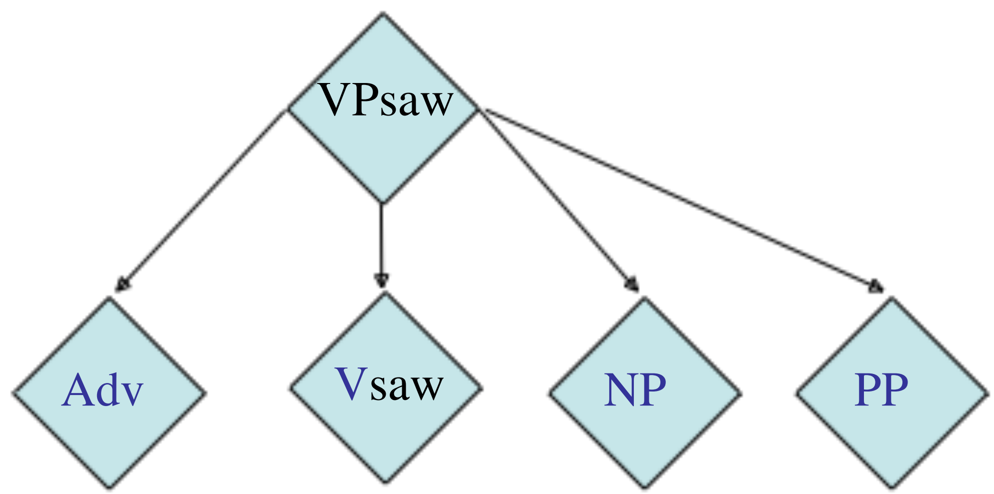
Charniak (1997) - in brief

- Generally similar to Collins
- Key differences:
  - Used an additional 30 million words of unparsed text in training
  - Rules not fully markovized: pick full nonterminal sequence, then lexicalize each child independently

Charniak (1997) - in brief

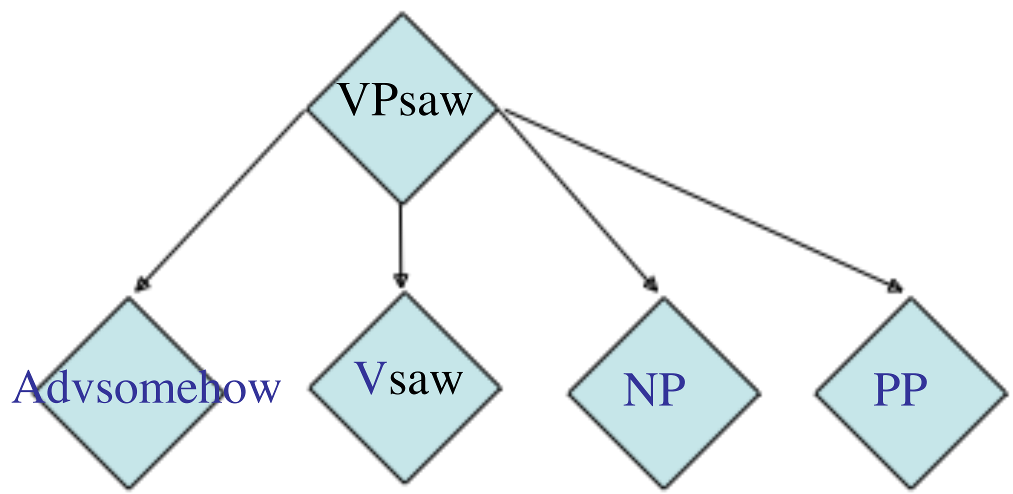
Charniak (1997) - in brief

VPsaw → Adv V NP PP



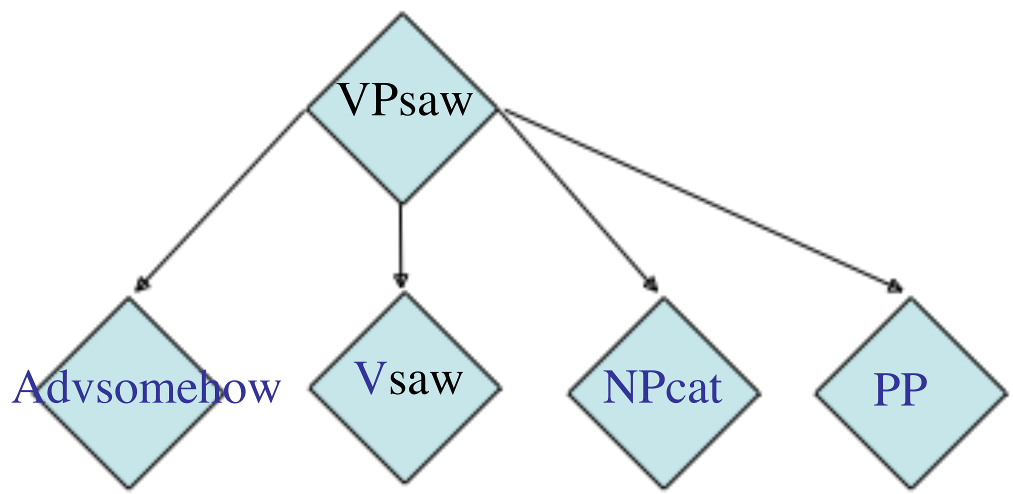
Charniak (1997) - in brief

p(somehow I VPsaw, Adv)



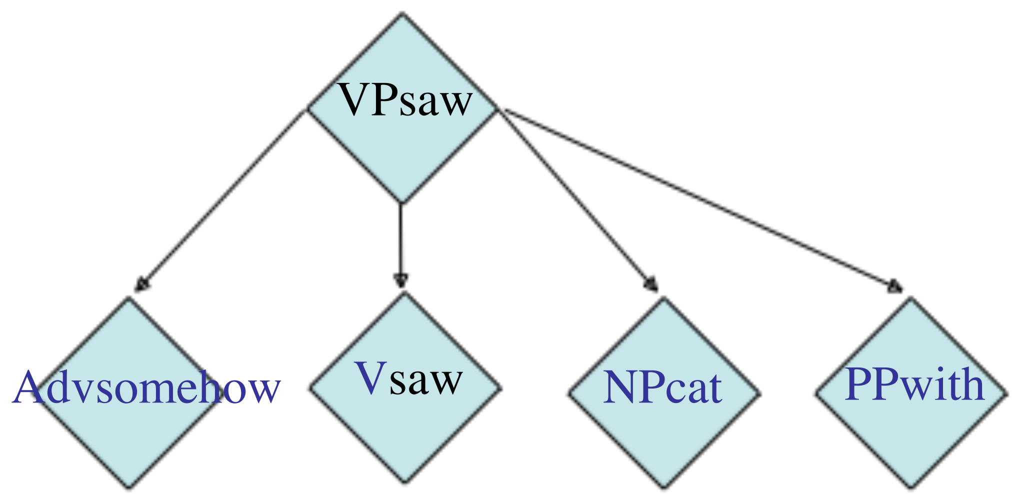
Charniak (1997) - in brief

p(cat I VPsaw, NP)



Charniak (1997) - in brief

p(with | VPsaw, PP)



Charniak (2000)

- Uses grandparents (Johnson '98 transformation)
- Markovized children (like Collins)
- Bizarre probability model:
  - Smoothed estimates at many backoff levels
  - Multiply them together
  - "Maximum entropy inspired"
  - Kind of a product of experts (untrained)

Comparison

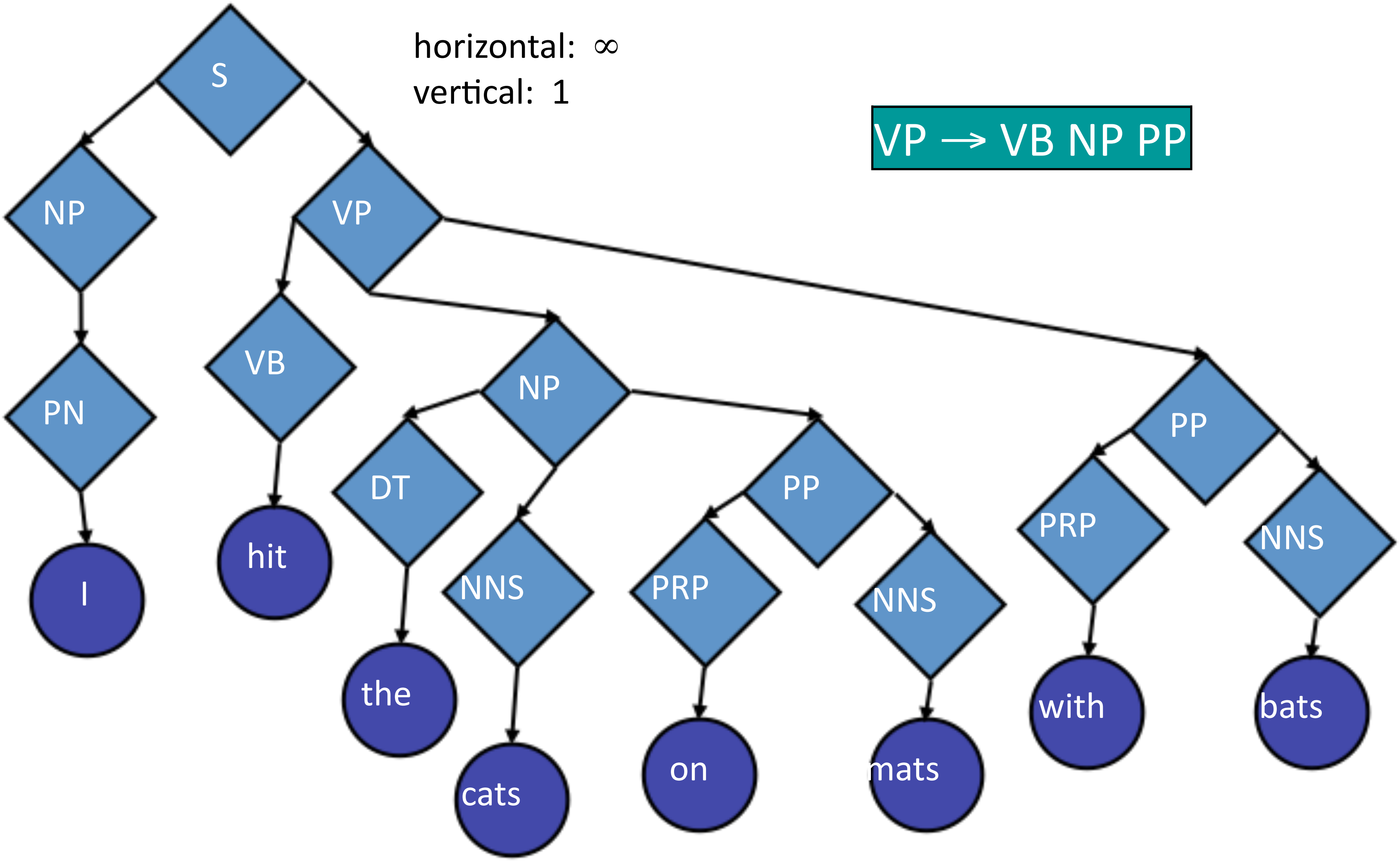
		labeled recall	labeled precision	average crossing brackets
Collins	Model 1	87.5	87.7	1.09
	Model 2	88.1	88.3	1.06
	Model 3	88.0	88.3	1.05
Charniak	1997	86.7	86.6	1.20
	2000	89.6	89.5	0.88

Klein and Manning (2003)

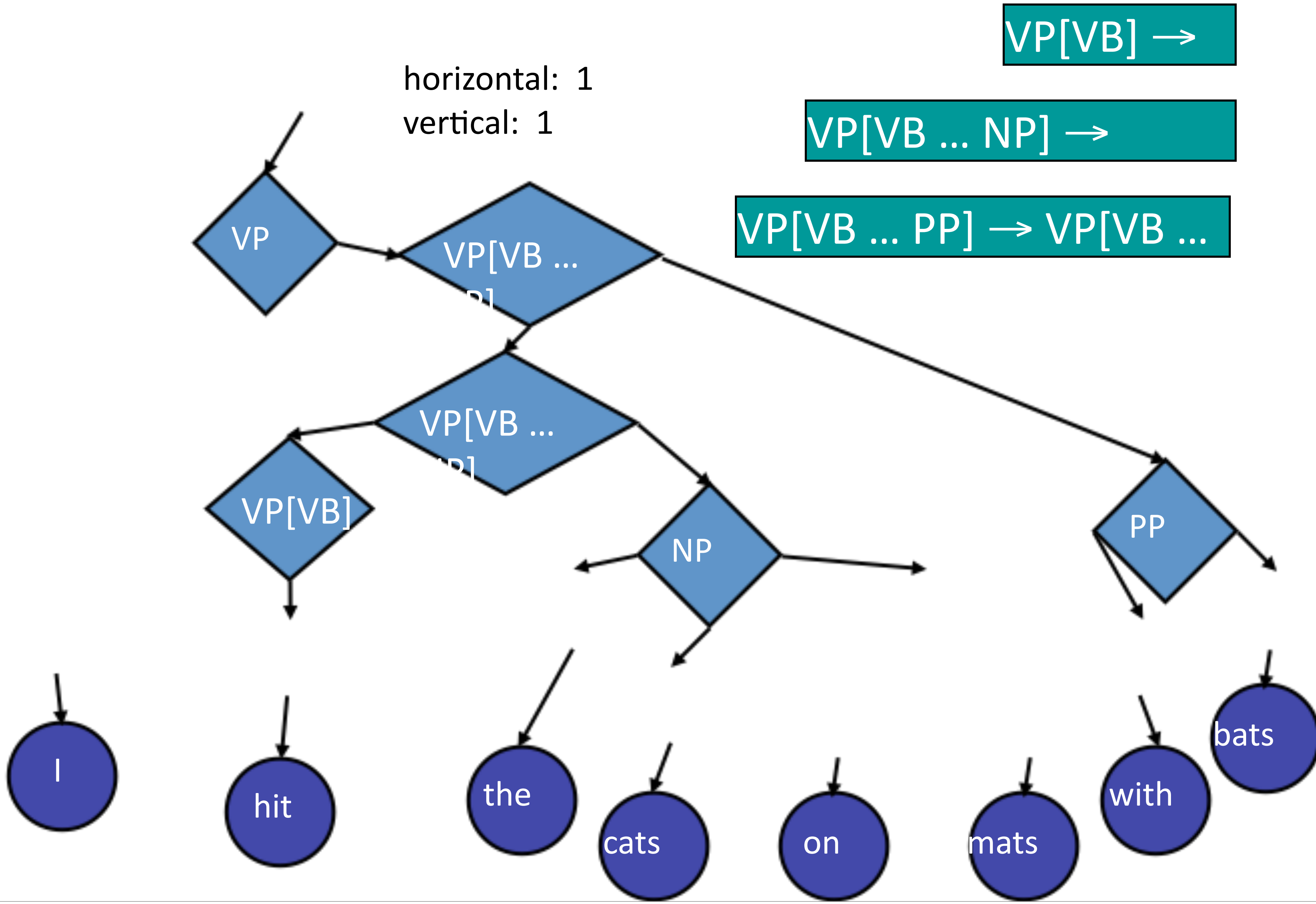
- By now, lexicalization was kind of controversial
  - So many probabilities, such expensive parsing: is it necessary?
- Goal: reasonable unlexicalized baseline
  - What tree transformations make sense?
  - Markovization (what order?)
  - Add all kinds of information to each node in the treebank
- Performance close to Collins model, much better than earlier unlexicalized models

Markovization

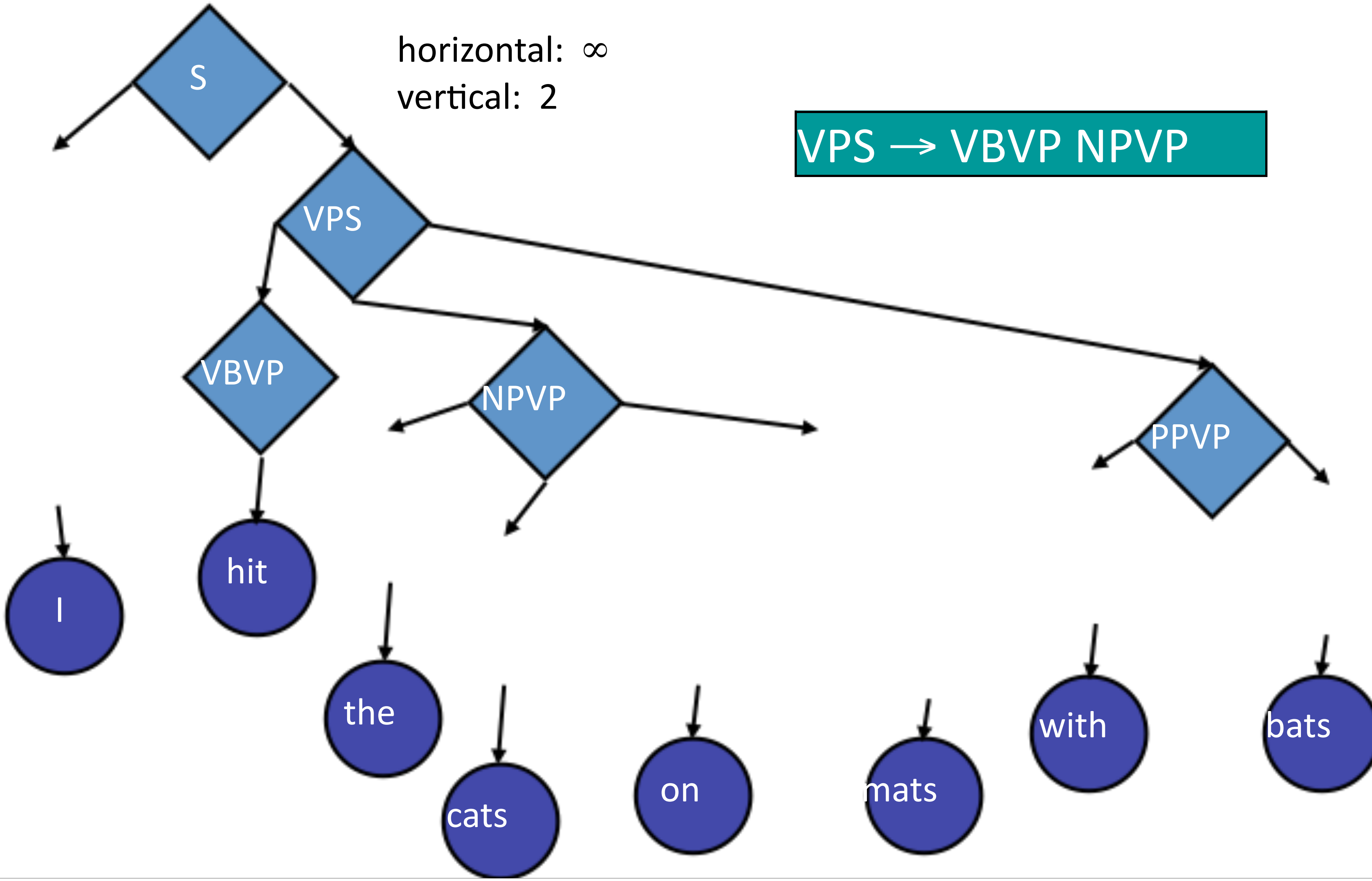
horizontal: ∞
vertical: 1



Markovization



Markovization



horizontal: ∞
vertical: 2

VPS → VBVP NPVP

Markovization

- More vertical Markovization is better
  - Consistent with Johnson (1998)
- Horizontal 1 or 2 beats 0 or ∞
- Used (2, 2), but if sparse "back off" to 1

Other Tree Decorations

- Mark nodes with only 1 child as UNARY
- Mark DTs (determiners), RBs (adverbs) when they are only children
- Annotate POS tags with their parents
- Split IN (prepositions; 6 ways), AUX, CC, %
- NPs: temporal, possessive, base
- VPs annotated with head tag (finite vs. others)
- DOMINATES-V
- RIGHT-RECURSIVE NP

Comparison

		labeled recall	labeled precision	average crossing brackets
Collins	Model 1	87.5	87.7	1.09
	Model 2	88.1	88.3	1.06
	Model 3	88.0	88.3	1.05
Charniak	1997	86.7	86.6	1.20
	2000	89.6	89.5	0.88
K&M	2003	86.3	85.1	1.31